Diploma Thesis

A GUI-based Interaction Concept for Efficient Slide Layout

Volker C. Schöch
vschoech@inf.fu-berlin.de

February 2003

Freie Universität Berlin
Institut für Informatik
Takustr. 9
14195 Berlin
Germany

Reviewers: Prof. Dr. Raúl Rojas
Prof. Dr. Klaus-Peter Löhr

Advisors: Dr. Markus Hannebauer
Felix von Hundelshausen

Abstract

A range of research systems have been developed for layout automation, but none of them suitable for business slide creation. There are, also, many efficient user interface techniques described in the literature, but until today none of them have been employed in popular presentation software. In this thesis, a field study was conducted that shows which problems arise when Microsoft PowerPoint is being used in a professional context. Based on the study, this research presents a new interaction concept for the efficient layout of business slides.

The *smart grid* is a technique that is used to numerically encode spatial constraints and to arrange *smart elements* on a slide accordingly. An intuitive interaction concept was developed that allows the efficient specification of smart grid constraints. The concept was implemented in a prototype, using efficient algorithms to achieve interactive response times. A dynamic programming algorithm is used to compute intelligent layout suggestions.

The prototype runs as part of a commercial product that enhances Microsoft PowerPoint with constraint-based layout support. In the product, a numeric constraint solver calculates the optimal smart grid layout for a given set of smart elements and placement constraints. A case study verified that the interaction concept as implemented in the prototype increases the efficiency for layout specification by about 20 % over PowerPoint alone, even for expert users.

Acknowledgments

First of all I wish to thank Prof. Dr. Raúl Rojas, since he made this work possible in the first place. I appreciate that Prof. Rojas supported my thesis despite the fact that its nature is not purely academic. Felix von Hundelshausen was my advisor in the work group of Prof. Rojas and I thank him for the good cooperation. I would also like to thank Prof. Dr.-Ing. Klaus-Peter Löhr for his willingness to act as a second reviewer.

Dr.-Ing. Markus Hannebauer and Arno Schödl, PhD, dedicated precious time and constant energy to support my work in all practical respects. Arno was always at hand with helpful technical advice on implementation issues, whereas Markus was readily available whenever I had one of the many questions that arise in the course of writing a diploma thesis. I want to thank both of them and their start-up company think-cell Software GmbH for providing the context and the environment to realize the present work.

The field study and the comparative evaluation were made possible by an unnamed business consultancy office in Berlin, whom I thank for their obliging cooperation and invaluable insights. Particular thanks go to Patrick Weese, a PowerPoint professional who spent many hours giving expert ratings and doing slides under observation.

Annika Hinze provided me with links for better writing skills. (I also borrowed some books from her and must not forget to give them back!) Miriam Busch gave out the LaTeX source code of her own thesis that served me as a working example and made it easy to start off writing. Thank you!

Winfried Schöch, Klaus-Bernd Schürmann, Amber & Till Weinhold and Armin Schöch read my work at different stages. Their valuable comments helped to put this text into shape and the reader has to thank them for an easy and understandable wording (if that is the case).

Last but not least a very special thank-you must be given to my dear wife Rasa and our little daughter(s) for their incredible patience, extraordinary support and unlimited understanding for my occupation throughout the last couple of months.

Contents

List of Figures

1 Introduction

> "*Layout* refers to the process of determining the sizes and positions of visual objects that are part of an information presentation. *Automated layout* refers to the use of a computer program to automate either all or part of the layout process. This field of research lies at the crossroads between artificial intelligence and human computer interaction."
>
> Lok and Feiner in [LF01]

This thesis deals with the human-computer interaction part of a new approach for automated layout. With the notion of *automated layout* I do not refer to graph layout or VLSI layout. These fields of research share some of the terminology used throughout this text, albeit with some subtle differences in meaning. The application domain of my automated layout approach are business slides, which to date are typically designed with Microsoft PowerPoint[1].

A field study in a business consultancy, which is detailed in chapter 2, showed that the tools provided by standard software like PowerPoint are not adequate for efficient slide design. For example, business slide layout typically consists of carefully aligned text boxes and arrows. Since the alignment tools provided by PowerPoint are merely pixel-based and do not create lasting relationships between shapes, setting up a completely aligned layout is a tedious task. In particular, when some more text or some new element needs to be integrated into an existing slide, the entire layout must be updated manually until all requirements are met. Standard requirements are: Certain shapes must have the same size, all text must have the same formatting, all shapes must be horizontally and vertically aligned with some other shapes.

The most regular, grid- or table-like layouts that prevail on typical business slides are apparently suitable for automated layout support. By *automated layout support* I mean the implicit or explicit specification of relations between drawing elements, and the dynamic maintenance thereof during an interactive layout process. A review of the available literature on automated layout is given in chapter 3. It turns out that a number of constraint-based systems for automated layout have been developed in research, but none of them have been successfully applied in practice. This observation is commonly attributed to the fact that those systems are very complex and therefore hard to use. However, the idea of user-specified layout constraints that are interactively maintained when the layout changes, appears to be an appropriate concept for professional slide design.

The start-up company *think-cell*[2] elaborated on that idea and is now developing an add-in for PowerPoint that realizes constraint-based layout automation. They already implemented an architecture, which allows software-driven interaction with

[1] All company and product names used in this text are trademarks or registered trademarks of their respective owners.

[2] My work is based on and part of a product from *think-cell Software GmbH*, Berlin, which is called *think-cell layout*. At the time of writing, it is in early development stage.
`http://www.think-cell.com`

PowerPoint – for example, to modify existing shapes or to add a new shape to a slide. Moreover, the people at think-cell developed and implemented a concept which they call the *smart grid*: A collection of horizontal and vertical *smart gridlines* that describes the position of each element on the slide.

Elements that are contained by the smart grid are called *smart elements*, because their locations are no longer pixel-based. Instead, each smart element is bound to a number of smart gridlines and moves along when the gridlines move. The effect being, smart elements that are bound to the same gridline, are aligned – and as long as the binding to the gridline is maintained, they stay aligned even when the over-all slide layout is modified.

At the time of writing, two essential parts of the software were not yet implemented: The numeric constraint solver that computes the "optimal" layout, and the user interface that is necessary to operate the smart grid. It is the theme of this thesis to realize an interaction concept that facilitates an easy and efficient constraint specification for smart elements and their placement in the smart grid. In chapter 4, the smart grid idea is explained in detail and an interaction concept is developed that leads to the specification of a user interface.

While the numeric constraint solver is still not in place, the existing architecture of the think-cell add-in for PowerPoint provided the opportunity to turn my user interface specification into a running prototype. The architecture of the prototype and some of the algorithms used for the implementation are discussed in chapter 5. Finally, chapter 6 evaluates the prototype by means of a comparative case study and quantifies the achievement in terms of the increase of efficiency over plain PowerPoint.

2 Field Study

> Jeff Hawkins (PalmPilot inventor)[3]: *"When I design products, I pretend to use them as much as possible before they exist. Meaning, I would put the wood prototype in my shirt pocket and carry it around all the time. If someone phoned, I'd take it out and pretend I was looking up something in the calendar or address book."*

In my case, to acquire a thorough understanding of the problems for which my work attempts to offer a solution, I can do even more than Hawkins with his wooden "prototype". Since the think-cell product is an extension to Microsoft PowerPoint, I could go and watch PowerPoint users in the target setting. A pilot customer of the think-cell company – a mid-sized consultancy office – provided an opportunity for me to do exactly that. The results are presented in this chapter.

2.1 Qualitative Aspects

Before we can even think about a *quantitative* evaluation we need some terminology and some basic understanding of the ongoing processes that involve Microsoft PowerPoint in the context of consultancies. A *qualitative* study provides the domain knowledge or the scale, which can be used to conduct a quantitative evaluation. In order to use time economically, I collected some quantitative data along with the qualitative study. However, the results are presented in distinct sections.

2.1.1 Questions and Methods

In order to design an appropriate user interface, I need to understand the user population, their environment, and their requirements. With regard to the user, the typical level of computer expertise is of special interest. Information about the environment includes typical technical equipment and the working conditions as far as interaction with software is concerned. Furthermore, a contextual task analysis [May99] provides information about the over-all workflow and thereby about the role that the intended software could play and the impact it may have. I used a range of complementary methods to collect and verify the required data.

Observation. While I was present at the consultancy's site, I watched the entire workflow and paid special attention to the people using Microsoft PowerPoint. Doing so, I collected notes which I later aggregated into statements.

Interviews. I asked people to take a few moments and answer those questions which I couldn't answer myself by simply watching – these are typically "why?" questions. If I had a hypothesis in mind, I tried to verify it by asking "right or wrong" kinds of questions. In some cases, I conducted longer and carefully prepared interviews to learn about some part of the workflow which I couldn't

[3] from `http://www.business2.com/articles/mag/0,1640,21,00.html`

cover by observation. An interview leads to in-depth information which cannot easily be generalized as it comes from a single source.

Surveys. Questionnaires are another means to verify hypotheses that came up from observation. As opposed to interviews, surveys support a generalization of a result because input comes from a range of sources. On the other hand, surveys cannot provide very much detail, because the need for statistical evaluation of the responses requires a uniform and restricted format of the answers. A survey can be used to weight the significance of results from observation and interviews.

Hands-on experience. To strengthen the feeling for the work and the problems and to understand the motivation behind some best practices, I tried to use Microsoft PowerPoint myself the way the people at consultancies do. When preparing my own presentations, I used graphical elements and layouts that I had seen before on slides from consultancies. Thus, I developed some skills in using the Microsoft PowerPoint tools for drawing, alignment, z-order, and so forth.

Representativeness of the data. Most observations, interviews and surveys were undertaken in one office of think-cell's pilot customer. In the course of two weeks, I was present at the pilot customer's site on five workdays, which translates to ten 6-hour shifts in the visual pool. The data collected at that site was supplemented and backed up by interviews with consultants from other consultancies. As it turned out, working techniques of different consultancies are extremely similar, thus providing reasonable argument to generalize my findings and conclusions.

2.1.2 User Profiles and Working Environment

There are strictly two types of Microsoft PowerPoint users in a consultancy. On the one hand, there are the *consultants* themselves. On the other hand, there are the visual assistants or simply *visuals*. The entirety of all available visuals at a point in time is referred to as the *pool* or *production*.

Consultants

While the focus and the value of the consultants' work lies in economic improvements of their customers' businesses, creating a convincing presentation as a basis for effective communication with the customer is almost equally high on their priority list.

Consultants do a lot of team work and also split up responsibilities for different aspects of a project and a presentation, respectively. However, the actual creation of slides is done privately. The consultant's typical workplace is constantly moving between the home office and one or more customer sites. She[4] has a laptop computer

[4]For ease of writing, I always refer to "the user" as *she* with no gender-specific meaning implied.

with usually only an external mouse attached, thus being bound to using a limited keyboard and a relatively small screen.

The consultant's day is determined by constant time pressure and overtime work. Besides personal notes and a calculator, Microsoft Excel is by far the most important tool for the consultant's work, followed by Microsoft PowerPoint. Microsoft PowerPoint is even preferred over Microsoft Word to create minutes of meetings, because PowerPoint slides are easy to show and to discuss in later meetings. Additionally, they can sometimes be put to further use in other PowerPoint presentations.

Visuals

The visuals typically are part-time student workers. They have a diversity of educational backgrounds which generally show no relation to their work as Microsoft PowerPoint experts. The expertise required to do the job in the pool is acquired on the job. For this reason, the actual skills in using Microsoft PowerPoint vary considerably among the visuals and relate fairly well to the duration of their employment. It is only after about two years of experience that a visual achieves top performance in using PowerPoint

The visuals share an office with a number of workplaces which they use in turn. There is no fixed relation between workers and places, meaning that only the most important software settings are changed according to personal preferences. The visual office is well equipped with up-to-date hardware including large screens and high-capacity printers. Approximately half a dozen workers share the same location. Due to this the computers' audio is generally turned off. The pool works in two 6-hour shifts, which are scheduled flexibly, according to each worker's personal needs and the demands of the consultancy's business.

2.1.3 Workflow

The workflow of a visual is part of a bigger picture, which is the consultant's workflow. Therefore, I first examine the workflow which represents a consultant's job with respect to presentation creation and then zoom in to the more specific workflow of a visual.

A Consultant's day

Customer communication and team communication together with the respective postprocessing make up the greater part of a consultant's day. When it comes to putting ideas and concepts into a presentation format, the consultant more or less works along the following lines (Fig. 1).

A. First of all, a **Story Line** is found. The story line represents the hypothesis or core statement of the presentation, which is the result of the preceding analysis. It is laid out on a single sheet of paper, as a story board of thumbnail slides and preliminary headlines.

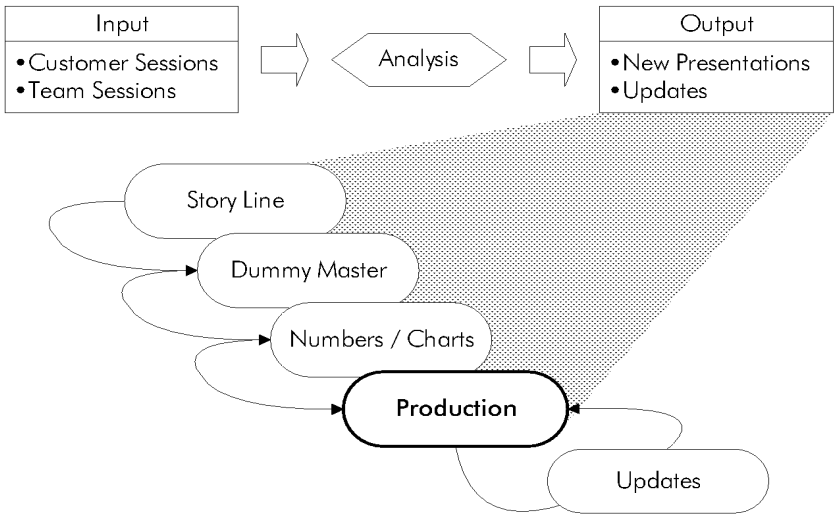


Figure 1: The consultant's workflow with respect to presentation creation

At this stage, the consultant looks for material and ideas from previous projects. This includes the retrieval of related slides from other presentations, talking to colleagues and searching the consultancy's knowledge management system (if available).

B. The next step is the **Dummy Master**, which is a collection of full size pages emerging from the story line. Besides some preliminary headlines, the sheets contain draft remarks and sketches. A preliminary agenda imposes some structure on the presentation.

   The dummy master and all the following steps are iteratively refined throughout the whole project.

C. Once the dummy master represents the train of thought that shall carry through the presentation, the consultant inserts the **Numbers** that are required to support the argument. Numbers in presentations are usually visualized as **Charts**, which are laid out at this stage.

D. Now the agenda is in shape, the numbers are available and the consultant has decided on how to visualize facts and concepts. At this stage, the dummy master is turned in to the **Production**, where the visuals take over. It is only now that PowerPoint comes into play. The consultant mostly works with pen-and-paper sketches and handwritten remarks on printouts of existing slides.

E. The first attempt to create a PowerPoint presentation from a dummy master, is never a complete success. Typically, the consultant's handwriting was not read correctly by the visuals, or – in view of the PowerPoint version of a slide – the consultant decides to explain some concept in a different way.

Therefore, any PowerPoint presentation gets back to the pool for **Correction and Update** at least once.

In the case of simple typing errors or numbers changing in a way that does not affect the slide layout, the consultants tend to do the necessary refinements in PowerPoint at their own desk.

A Visual's Day

The workflow of a visual assistant (Fig. 2) is more specific and uniform than the consultant's workflow. Work is received by the production pool in batches of slides, typically by telefax. Entirely new slides are created as freehand sketches. Alternatively, consultants use printouts of slides (not uncommonly from a range of different presentations) and modify them manually with pencil and correction tape.

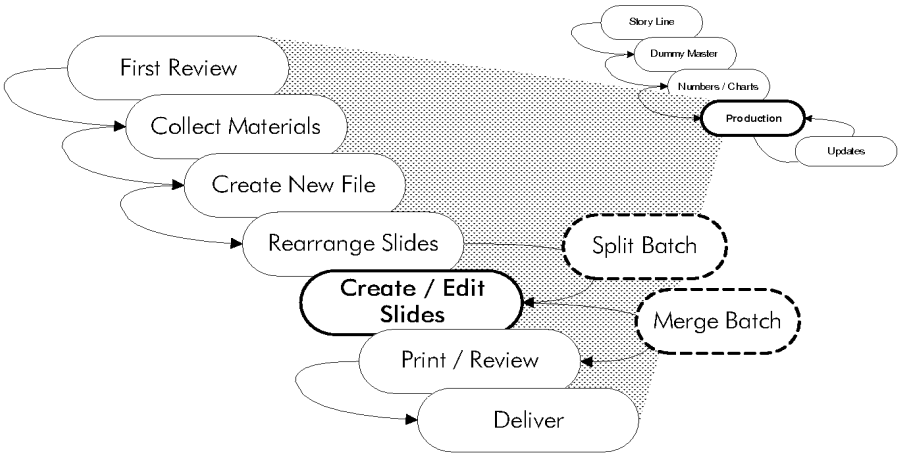


Figure 2: The visual's workflow as part of the consultant's workflow

A. The first step is to **Review** the batch of paper in order to get an idea of the type and amount of work. If there are a lot of very difficult slides, a relatively unexperienced worker might give the batch to a more experienced one.

B. Next, all required **Materials are Collected** into a local working folder. This includes PowerPoint files, Excel files, images and so on.

C. An existing presentation or template is opened in PowerPoint and the Save As... feature is used to **Create a New File**.

D. If the new presentation is based on an existing one, then at this stage the **Slides are Rearranged** to match the new presentation. Slides that are no longer needed are removed and existing slides from other presentations are copy-pasted to their destination.

E. If the visual estimates that a batch might take a long time and if there are other visuals available, at this time the visual may **Split the Batch** into several pieces so that it can be worked on in parallel. To do this, the current state of the presentation is duplicated to the other visuals' machines so that each visual works on a local and private copy of the PowerPoint file.

F. After these preparations, the batch is taken slide by slide. **New Slides** are inserted and existing slides are modified, making this part of the workflow the one with the most intensive PowerPoint interaction. This step is determining the over-all amount of time that is required to process the batch.

G. If the batch was split up between two or more visuals, at this stage all files are copied to a single computer and the **Batch is Merged** into a single file, using copy-paste within Microsoft PowerPoint.

H. Then, a **Printout** is created from the entire file. It is **Reviewed** and compared to the scribbles turned in by the consultant. Last changes and corrections are performed and the affected slides are printed again for review.

I. Finally, the file is **Delivered**. More specific, the file may be sent to the consultant's site by email or some other means of remote file sharing, and the consultant is notified.

2.1.4 Working Techniques

During the observations in the visual pool, I took notes on a large number of working techniques. In this section I present only those techniques that are most important to understand the workflow involving Microsoft PowerPoint and to explain the potential of a better user interface.

Yellow Pad. The *Yellow Pad* is the consultant's sketch pad, made of heavy yellow paper with light marks on it. The marks help with sketching, but they do not photocopy or telefax.

New slides are first sketched out on the yellow pad. Due to bad handwriting and sketchy drawings, even an experienced visual needs some time to understand the wording and the graphical elements of a slide. Visuals say that they cannot decipher about 10 % or more of the manuscript, in which case they mark the word in question and continue without interrupting the workflow.

Charts are usually reproduced from the manuscript just as any other figures – meaning that the consultant copies all numbers from Excel sheets to the yellow pad and the visual copies all numbers from the yellow pad to PowerPoint. The visuals generally do not have access to the underlying data sources, for practical reasons as well as for reasons of confidentiality and data integrity.

However, pure yellow pad scribbles account for only a small fraction of PowerPoint work. For the most part, existing presentations are modified, or consultants

take slides from different PowerPoint files and adapt them for a new presentation, using correction tape and handwritten remarks.

A Consultant's Tools. Apart from communication devices – telephone, cellphone, telefax, email, organizer – there's only a handful of tools which are essential to the consultant's work: Microsoft Excel, Microsoft PowerPoint, a calculator, a pen, a personal note pad, a yellow pad and correction tape.

The entire project related communication with team members as well as with customers is based on PowerPoint slides and Excel sheets. Both kinds of data evolve continuously throughout the course of a project. Presentations of intermediate results are usually based on the same, iteratively refined PowerPoint file. The final version of the file at the same time serves as a documentation that is handed over to the customer when the project ends.

In Microsoft Excel, consultants mostly use features for numeric aggregation (sum, average, etc.). They also make heavy use of live links between sheets and even between Excel files, but apparently they never create live links between Excel sheets and PowerPoint presentations. Excel sheets are sometimes used to hold large amounts of text, which requires all kinds of tedious work like manual line wrapping, formatting and merging of cells, because Excel is not prepared for managing entire paragraphs of text. On the other hand, the elaborate Excel features to create charts are almost never used, because charts are only required for presentations and it is left to the visuals to create those charts in PowerPoint.

Basic Slide Layout. There is a common standard to which most slides produced in a consultancy adhere. Some of these elements are also marked on the yellow pad blanks. The most common elements, their use, formatting, and meaning is indicated in figure 3.

It is striking that consultants typically put loads of information onto a single slide. Students in academia are told to use font sizes no smaller than 18 points. Consultants use font sizes *no bigger* than 18 points and on dense slides go down to even 10 points. However, consultancies as well as academics agree on the general rule that there should be no more than three different font sizes on a single slide.

Consultants make extensive use of graphical elements like boxes, lines, bubbles and a variety of different arrows and pointers. They also like to visualize recurring concepts by a thumbnail of the figure which first explained the concept. Depending on the context and purpose, consultants use detailed charts based on actual numbers, or symbolic charts that convey a certain tendency or idea.

Best Practices. The PowerPoint experts in the visual pool developed a range of best practices, which are used to solve a certain problem in a certain way. Typically, so-called best practices provide an indication of missing or unusable features in the software being used. Some of the best practices are well-known and shared by everybody involved, others are more of a personal habit. For this reason, visuals

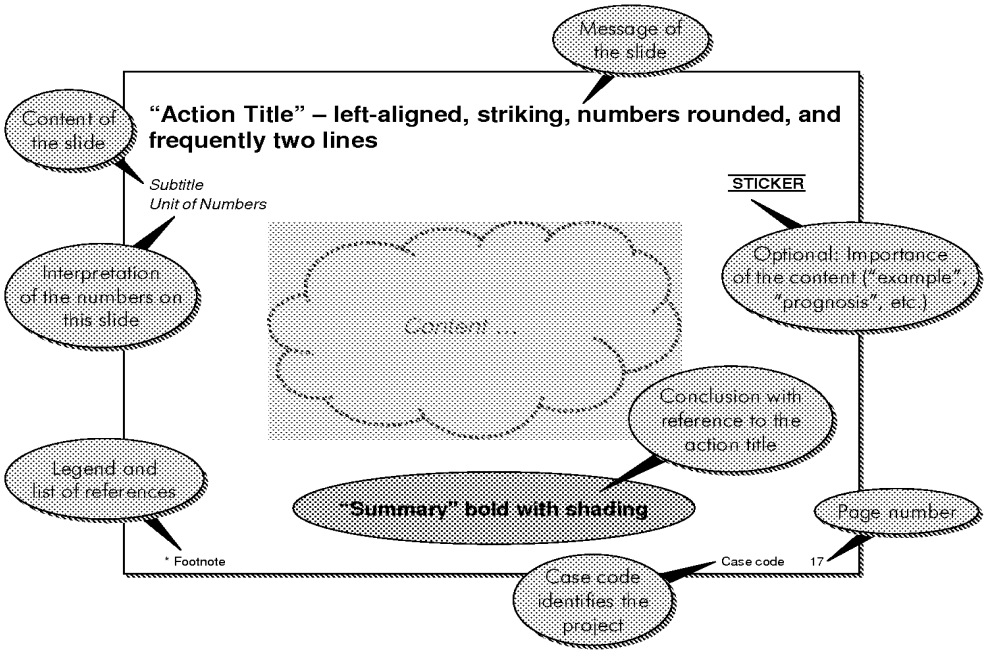


Figure 3: Basic elements of a consultancy slide

generally do not appreciate it when consultants work on the slides themselves. They reportedly prefer even minor modifications to be sent to the visual pool so that the fragile composition of a complex slide is not broken.

2.1.5 Deficits of Microsoft PowerPoint

A large number of the working techniques that I observed throughout the field study, turned out to be workarounds for problems with the usage of Microsoft PowerPoint. In this section I present selected problems that could be remedied by layout automation.

A. PowerPoint Automation

(1) PowerPoint supports automatic resizing of text boxes to fit the text. Since a text box is usually aligned with other shapes on the slide, this feature tends to distort the slide layout and is therefore disabled.

(2) PowerPoint supports automatic adaption of the text size to fit a text box. However, only the text size of the over full text box is affected. Since generally a slide should have uniform font types and sizes, this feature is always disabled. If required, text sizes of all elements on the slide are adapted manually.

(3) PowerPoint supports automatic text wrapping, but the algorithm produces unusable results: Syllabication is neither supported at all, nor are

hard spaces and potential hyphens as they are known in Microsoft Word. Instead, sometimes a line break occurs at an arbitrary position within a word and without a hyphen. This is not acceptable and visuals prefer to disable automatic wrapping all together and insert all line breaks manually.

B. Alignment

(1) The PowerPoint grid is not stable through multiple runtime instances of PowerPoint. If a presentation is reopened, elements that were previously aligned with the grid frequently happen to be just slightly off. In this case, modifying a complex slide becomes a nightmare. The only cure is to select all elements of the slide and push them back into the grid, before changing or adding any shapes.

(2) When Snap to Grid is active and a shape is moved with the mouse, it is unclear which point of the shape actually is aligned with the grid. If a shape is moved with the cursor keys, alignment depends on the direction of movement.

(3) The number of magnetic guidelines is constrained to 8 per dimension. Since guidelines relate to an entire presentation (as opposed to a single slide), there are only 8 vertical and 8 horizontal positions available. This is not enough when a presentation contains numerous complex arrangements.

(4) The guidelines and the grid are hard to see on a slide; they are displayed as very small dots in light gray. This is unnecessarily tiring for the visuals' eyes.

(5) Even carefully aligned elements that seem to fit perfectly in normal view, frequently turn out to look distorted in the presentation view or printout. The only reliable way to avoid these kinds of problems, is to view the drawing with the maximum zoom factor of 400 %.

(6) Generally, it is possible to resize multiple shapes synchronously by just doing a multi selection. A strange exception to this principle exists: Lines can only be kept aligned during resizing, if they are grouped beforehand.

(7) The alignment tools always move the shapes. There is no function to align, for instance, all right edges of a couple of text boxes, and leave all the left edges where they are (effectively resizing the shapes).

C. PowerPoint Charts

(1) Although the Microsoft applications are controlled by the means of a graphical user interface, the principle of direct manipulation is not always applied. In the case of charts, it is not possible to change the height of a column or the size of a slice with the mouse. To achieve these

adjustments, the user must go back to the data table and change the underlying numbers.

(2) PowerPoint charts provide some means to label segments of a column, for instance. However, the placement of labels does not match the requirements of the consultants. The consultants have a set of conventions to improve legibility even for labels of very small segments. PowerPoint doesn't support these standards and therefore visuals generally place labels manually (meaning redundant data input and thus another source for errors).

(3) Furthermore, any text placed within a chart cannot be sufficiently controlled by the user. Font size and line wrapping are adapted by PowerPoint, leading to unpredictable results. This is another reason why visuals turn off any automatic text in charts and manually create an extra text box for every single label.

(4) It is hard to scale charts exactly to a certain size. If the chart object is resized as it appears in PowerPoint, distortion occurs. The chart can only be resized undistortedly in chart editing mode. In this mode, however, the relation between the chart size and the underlying slide is not clear, because the chart is magnified for editing.

D. Tables

(1) Table columns in PowerPoint cannot be assigned a precise width; they can only be resized using the mouse, resulting in an irregular layout.

(2) Just as with any other shapes, line wrapping doesn't produce satisfying results. In contrast to other shapes, automatic line wrapping cannot be disabled in tables. The same reoccurs in the automatic resizing of table rows.

(3) For these reasons, the existence of the PowerPoint table feature is simply neglected in the visual pools. Visuals generally build table-like arrangements from single text boxes. The disadvantage is obvious: When first creating the table, multiple attempts are required to determine the optimal size for the text boxes to fit the page. Inserting new rows or columns later is almost impossible.

E. Keyboard use and navigation

(1) There are many situations in which a desired effect cannot be achieved just by using the keyboard. In these cases, using the mouse is obligatory. A prominent example for this is the change of the focus from one text box to another, which is frequently needed because almost anything on a slide – including tables and chart labels – is realized as text boxes. The need to constantly switch between keyboard and mouse significantly reduces efficiency and accuracy of PowerPoint work.

(2) As mentioned above, extremely high zoom factors play an important role for slide layout. However, in a zoomed view it is difficult to navigate the slide. The only way to move the small visible part of the slide is to use the mouse and interact with the scroll bars. This is tedious because it is not possible to navigate both dimensions in a single interaction and the scroll bars are not helpful to find a specific area of the slide. Navigating the view with the keyboard is not possible at all.

(3) Also, frequently used functions like align, distribute and the like are only available as toolbar buttons and cannot be accessed using the keyboard.

2.2 Quantitative Aspects

In section 2.1, we have developed some terminology and a general understanding of the matter that is subject to this work. In this section, the problems that the qualitative study revealed are understood as potential opportunities for improvement. An attempt is made to quantify the potentials.

2.2.1 Questions

The quantitative study was designed to rank and quantify the impact of different PowerPoint-related activities on the over-all presentation creation process. Activities were grouped into the following categories:

A. **Data Entry** – typing in text and numbers from the manuscript into text boxes and data sheets.

B. **Slide Layout** – creating and arranging frequently-used visual elements like boxes, call-outs, lines and arrows.

C. **Chart Design** – creating a specific chart type, including layout of labels. Chart design is further categorized into chart types (for some examples, see figure 4):

(1) Bar charts (horizontal bars)

(2) Column charts (vertical bars)

(3) Waterfall charts (an exploded variation of column charts)

(4) Scatter charts (points in a two-dimensional area)

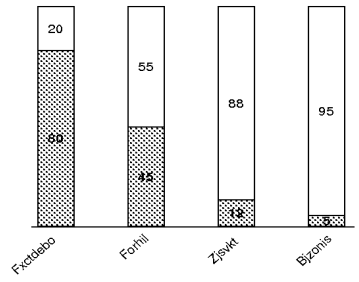
(5) Gantt charts (project time lines)

(6) Orga charts (organizational/hierarchical structures)

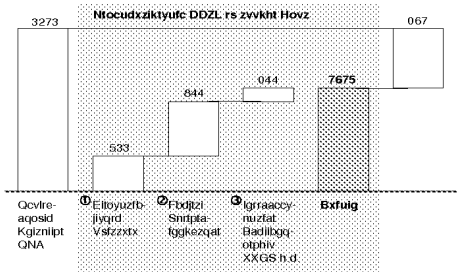
(7) Line charts (suggesting continuity of measure)

(8) Pie charts (visualizing a partition)

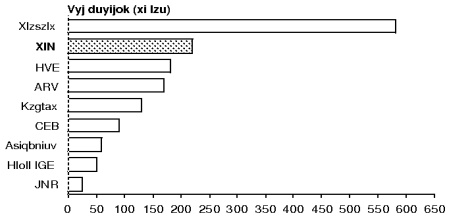
D. **Custom Drawings** – figures and shapes that are too specific to fall into any of the other categories.



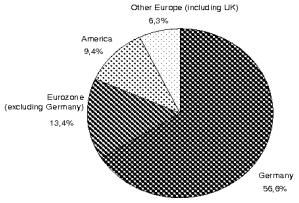
(a) Column chart



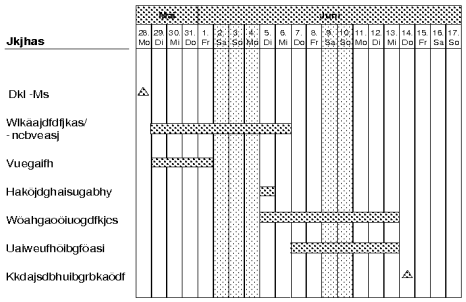
(b) Waterfall chart



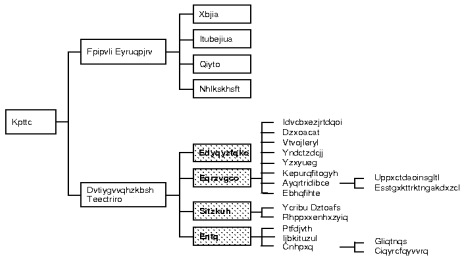
(c) Bar chart



(d) Pie chart



(e) Gantt chart



(f) Orga chart

Figure 4: Examples for some common chart types (from original slides, text contents intentionally scrambled)

Because this work is focussing on software-based solutions, activities that do not imply PowerPoint interaction are not taken into account. This includes the following typical stages of the workflow:

- Review – browsing through all slides before and after the PowerPoint work.
- File system interaction – searching, moving and sending data files.
- Interpretation of manuscripts – manuscripts are usually hard to decipher and the layout specification is often ambiguous.
- All kinds of interruptions – telephone calls, abrupt switches to another task with higher priority, and the like.

The intention of the quantitative study is to clarify which PowerPoint-related problems have the most impact on the over-all working process. At the same time, data emerging from a quantitative evaluation of current PowerPoint use provides a baseline for comparative follow-up studies. A case study that evaluates the impact of my work is presented in chapter 6.

2.2.2 Methods and Data Validity

While data collected on-site and during the actual process is most authentic, it takes too much time to collect enough data for a valid statistic evaluation this way. Therefore, mass data was collected by indirect methods off-site, while on-site observation was used to check the results for plausibility.

Again, data was largely collected from a single consultancy office. Comparison with sample presentations from other consultancies showed that – with little deviation of the absolute numbers – the general layout and the use of visual elements and chart types is sufficiently similar across different consultancies to generalize the findings of the study.

Live Timing. As a first approach to quantitative data, I did on-site user observation and live timing of PowerPoint related activities. For each item, some specifics including the type of activity and the skill level of the performing user were recorded along with the actual time required to perform the task. Moreover, for each tuple of data the affected slides were kept to allow a comparative study later.

Expert Ratings. To collect a statistically relevant amount of data in reasonable time, I decided to rely on off-site expert ratings. Together with an experienced visual, I went through 6 entire presentations that were selected as a representative sample, with a total of 371 slides. For each slide, I had the expert estimate the total time required to create the slide as well as times for text input and single elements like drawings and charts.

Data collected from expert ratings was backed up with a random sample of other experts' ratings and with live timings. The verification confirmed the estimated times to be plausible and accurate.

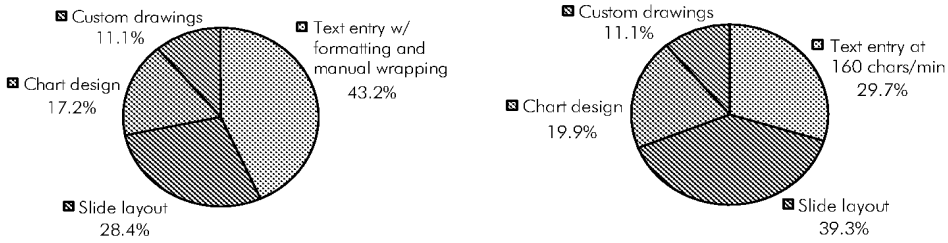
Analysis and Counting. As another off-site method to efficiently collect larger amounts of data, I went through 9 presentations (567 slides) that were rated to be representative. I categorized graphical elements and chart types and counted the occurrences.

Moreover, I put aside those slides that appeared especially difficult for computer support. The sample of difficult slides helps to validate new layout concepts and to clearly define the requirements for a better solution.

Interviews. Just as for qualitative data, interviews are an appropriate method to access hard-to-measure quantitative data as well. In this case, personal interviews were the only way for me to collect data concerning the consultants' point of view.

2.2.3 Proportionate Times of PowerPoint Interaction

Based on the data from expert ratings, the relative times spent with the four categories of PowerPoint interaction are shown in figure 5(a). In this chart, the slice for *text entry* takes up a strikingly large fraction of 43.2 %. Further evaluation of the data reveals an estimated average typing speed of **110 chars./min**, which is extraordinary slow.



(a) Time exposure (raw expert ratings) (b) Time exposure (revised)

Figure 5: Proportionate times of PowerPoint-related activities (based on expert ratings for 371 slides from 6 different presentations)

In order to understand obviously inaccurate numbers, I double-checked the actual typing speeds of a random sample of four different visuals. It turned out that all four of them were typing far **beyond 200 chars./min**, supporting similar findings of around 230-280 chars./min for frequent keyboard users who are not touch-typing.

The mismatch between actual typing speeds and estimated typing speeds can be explained if we go back to section 2.1.5, which explains some of the most annoying deficits in the Microsoft PowerPoint user interface. The items A3 and A2 explain that line wraps, syllabication and font sizes are handled manually by the visual, because the results of PowerPoint automation are generally unacceptable. Similarly, item C2 states that labels in charts are sized and placed manually, too.

With these best practices in mind, the estimation for text input appears plausible. However, the resulting partition is not helpful to analyze the potential impact of an automated layout tool. To correct the underlying error in the attribution of time to activities, I am assuming a moderate typing speed of **160 chars./min**. This assumption takes into account that typing of short paragraphs or single words – which is characteristic for PowerPoint slides – does not reach full speed in comparison to an uninterrupted passage of text.

Based on this assumption, the slice for *Data Entry* shrinks to 29.7 %, which is still a large fraction of the over-all PowerPoint interaction time but more plausible than the original estimate. The difference of 13.5 basis points is attributed to manual text wrapping, adaptation of font sizes and placement of labels. Based on the observations, these activities contribute mainly to *Slide Layout* (80 % of the difference) and partly to *Chart Design* (20 %).

The resulting distribution is shown in figure 5(b) and the potential contained within is impressive: Although I take for granted that there is no way to provide ad-

ditional software-based support for *custom drawings* and *text entry*, the remaining fraction makes up for almost two-thirds (59.6 %) of the over-all PowerPoint interaction time. In effect, this means that any efficiency gains in *slide layout* and *chart design* could have a major impact on the presentation creation workflow.

2.2.4 Frequency of Different Chart Types

The counting of slides and their composing elements was mainly focussed on chart types, including chart types that are not explicitly supported by Microsoft PowerPoint. My findings (Fig. 6) make a strong point that a focussed improvement for only a few chart types could have a major impact on the over-all slide creation workflow: Only three different chart types make up for three-quarters (76 %) of all charts used throughout an average presentation. In addition, the three most frequently used chart types – bar charts, waterfall charts and column charts – bear great structural similarity, thus further encouraging a focussed approach.

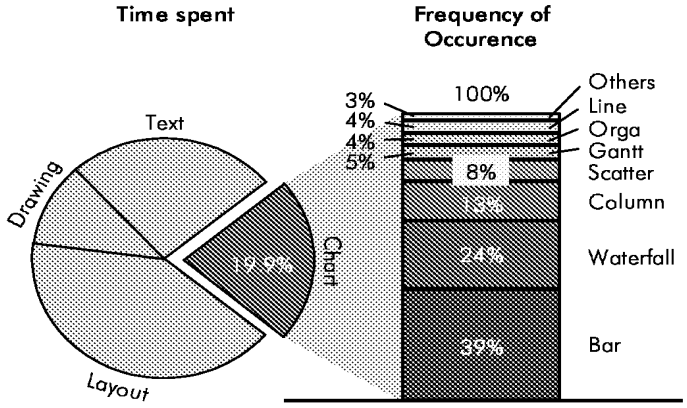


Figure 6: Relative frequency of chart types in an average presentation (based on 567 slides from 9 different presentations)

2.2.5 Quantifying the Consultant's Workflow

Concluding my quantitative findings, this section presents some numbers that represent the consultant's point of view. The consultant's workflow is explained in section 2.1.3 as part of my qualitative results.

The origin of a presentation is a yellow pad scribble. There are roughly two categories of slides: (a) Visualization of numbers and (b) explanation of concepts. Slides from category (a) generally do not comprise much more than one or two charts. The charts are quickly sketched out and numbers are written down from an Excel sheet. An entire slide of this kind is created on yellow pad within 10 min on average and at the first attempt is usually suitable to be sent to the pool.

On the other hand, conceptual slides (b) are much harder to create, because the consultant has to put down her ideas in telling geometric forms and concise

wording. Often, it takes two or more attempts and up to half an hour of time to come up with a utilizable sketch. Slide layout obviously is not only an issue in Microsoft PowerPoint but also on the yellow pad.

Every slide that is designed by the visual pool requires some overhead in communication. I presumed that the additional necessity for communication would be an incentive for the consultants to do some of the easy PowerPoint work at their own desk. In fact, consultants frequently use PowerPoint themselves, but obviously for other reasons. The additional efforts for communication with the visual pool are reported to be irrelevant in the light of the over-all workflow.

A final number: The visual pool is a resource which is not only valuable but also expensive. It is estimated that the creation of an average slide costs about €20 in terms of wages and non-wage labor costs (i. e., social insurance contributions, office room and equipment, etc.).

2.3 Field Study – Résumé

My field study yielded three important insights.

- The preparation of presentation slides plays an important role in the workflow of a business consultancy. A substantial fraction of the budget in terms of time and money is spent with slide design in Microsoft PowerPoint.
- Only experts with years of experience are able to use Microsoft PowerPoint efficiently. However, even experts are far from optimal performance, because they have to spend a lot of time to work around problems in the PowerPoint user interface.
- Slide layouts for business slides are uniform and regular. These characteristics provide a promising setting for a computer-supported layout approach.

In the next chapter, we are going to see what literature has to offer on these topics. We will find that to date neither research nor any commercial products have addressed the problems that have been revealed in my field study. Therefore, in the upcoming chapters of this paper, I'm going to present a new approach that substantially increases the efficiency for business slide layout with Microsoft PowerPoint.

3 State of the Art

In this chapter we identify related work and discover their relevance to the application of computer support for the interactive layout of business slides. Primarily, there are two different aspects to explore: On the one hand, there are theoretical and algorithmic considerations regarding the complexity, the power and the applicability of techniques that create precise and esthetically appealing output from an imprecise and incomplete specification. On the other hand, in the given setting I must provide means for the user to interact with the layout engine and must use input from direct manipulation, where possible. This leads to considerations of user interfaces (UI) and usability both in the special context of computer supported design and in the broader context of WIMP[5]-based applications in general.

I will first present an overview about menu-based interaction techniques as found in the literature. Next, the alignment tools that are most commonly found in popular software will be discussed. The rest of the chapter deals with research that has been conducted on computer supported layout.

3.1 User Interfaces for Layout Specification

Human-computer interaction (HCI) is a growing field of research that comprises psychological, physiological and technical factors and explores their relationship to one another. As with psychology in general, it is characteristic for the field of HCI that there are only few widely agreed-on facts available to design and measure "usability"[6]. Instead, research in HCI typically proposes incremental improvements to existing interfaces or examines the potential of entirely new UI devices. There is still a main focus on the traditional setup of desk, keyboard, mouse and screen with a graphical, window-based interaction paradigm. The latter is also the environment in which business slides are created.

3.1.1 Of Mice and Menus

The user interface that is the subject of this thesis is built on top of Microsoft PowerPoint and must integrate smoothly with the existing behavior and UI widgets. Since most interaction modes – including toolbar buttons and shortcut keys – are already in use to operate plain PowerPoint, there are mainly two interaction modes left to implement an additional UI: Direct manipulation of some on-screen visualization of constraints, and additional menus to control more abstract options.

Direct manipulation is agreed-upon to be the interaction mode of choice, where possible. This is because direct manipulation requires nearly no cognitive overhead and integrates perfectly with the user's mental model of, in my case, slides and drawing objects, which makes the interaction feel "intuitive".

However, there is some desirable functionality that cannot be easily mapped onto direct manipulation, because it goes beyond selecting and moving on-screen

[5] WIMP stands for *windows, icons, menus,* and *pointers* (e. g., [DFAB98])
[6] The term *usability* is briefly explained in section 4.2.1. For a detailed definition see [DFAB98].

objects. Examples include the specification of abstract constraints, text and color formatting options, and special settings for complex elements like charts. This functionality must be handled by menus. Boritz [Bor90] provides an overview of traditional menu techniques (and a title for this section).

Figure 7: A pie menu as seen in *The Sims* from Maxis (screenshot from [Hop])

Pie Menus and Variations

A particularly interesting variant of the traditional linear menus are *pie menus*. In pie menus, the menu items are laid out in a circular fashion around the current mouse pointer location. The distance for the mouse pointer to travel in order to select a menu item is equal for all items, and is almost minimal. The responsive areas of menu items in a pie menu have the shape of a wedge. They are especially easy to hit, because with increasing distance from the pie center the wedge grows much larger than traditional menu items.

Callahan and Shneiderman [CHWS88] were among the first to compare the performance of pie menus and linear menus. They found a statistically significant difference between menu type performance. Users were approximately 15 % faster with the pie menus and errors were less frequent than with linear menus.

Hopkins [Hop91, Hop] took the pie menu concept and implemented pie menus for a range of widely used applications, including the game *The Sims* (see Fig. 7). Pie menus for the Mozilla web browser [OR] are another prominent example.

The pie menu concept was broadened by the notion of *marking menus*. Marking menus refer to an invisible pie menu that allows menu item selection by drawing *marks* in different angles. Marking menus are non-obtrusive in the sense that they

Principle	Refinements
"Maintain visual context"	• Display only the labels (no backgrounds) • Violate geometric layout of pie wedges • Make labels symmetric
"Hide unnecessary information"	• Hide parent menus
"Support skill development using graphical feedback"	• Use eight menu items • Use compass star with menu center • Show idealized marks

Figure 8: Design refinements on marking menus as suggested by Tapia and Kurtenbach [TK95] (annotated)

do not cover the visual context of an interaction. Yet, they are easy to learn, because *not* drawing a mark, triggers a normal pie menu as a reminder of the menu item layout. Kurtenbach and Buxton [KB94] studied user learning and performance with marking menus and concluded that marking menus are a very effective interaction technique, as long as the list of items does not dynamically change. They suggested that marking menus would be an appropriate technique for user interaction in many popular software such as Microsoft Word.

Based on existing implementations of marking menus, Tapia and Kurtenbach [TK95] suggested some refinements and principles on the appearance and behavior of marking menus (Fig. 8). Friedlander *et al.* [FSM98] suggest another variation, which they call *bullseye menu*: A bullseye menu is a series of concentric circles divided into sectors. The effect being, the wedges of the normal pie menu are further divided so that more menu items can be presented in a single menu.

Other Menu Types

A great deal of research has gone into different ways to enhance traditional menus. Cornelison [Cor] as well as Sears and Shneiderman [SS94] compared user performance for static menus with performance for *split menus*, which group frequently used items in a dynamic section of the item list. They found that for specific applications and user populations split menus perform significantly better. We find this technique in the Microsoft Windows XP Start menu, where recently used applications are presented first.

Guimbretière and Winograd [GW00] developed a concept they call *FlowMenu*, which effectively is a hierarchical pie menu with submenus shown in the same place like the parent menu. Multiple stage menu selections can be performed in one interaction, allowing the use of a non-visible menu similar to marking menus. That same idea is pursued by Pook *et al.* [PLVB00] who present what they call *control menus*. In this case, a single gesture suffices to select an operation and make a choice from a continuum of possibilities, such as selecting the zoom option first and then setting the desired zoom factor.

All menus that have a visual representation have in common that the user must divide her attention between the context of her work and the menu layout. Harrison *et al.* tried to reduce the distraction that stems from visual interference between the

working context and the appearance of a menu by using transparent layered user interfaces [HIVB95]. This approach was repeatedly evaluated in different contexts [HKV95]. Results indicate that visual interference in fact impairs user performance and that transparent user interfaces can help to maintain context during the interaction.

3.1.2 Alignment Tools

Most of the layout activities that are required to design a business slide can be subsumed under the notion of *alignment*. Geometrically aligned elements give a clean and concise impression of a presentation and guide the eye to focus on the message it conveys. There is a range of UI widgets that support aligned layout.

Regular Grid. The regular grid as an aid for freehand drawing was first suggested by Hurlburt [Hur78]. It effectively prunes the space of possible shapes to those composed of lines whose endpoints horizontally and vertically align with an integral multiple of the grid's base unit. This restriction facilitates the construction of geometrically regular and aligned shapes with quick and rough mouse movements. However, the regular grid has serious limitations. For instance, it is impossible to create an equilateral triangle using a regular grid, and in order to center-align two shapes, care must be taken that both shapes span either an even or an odd number of grid units. Yet, the regular grid is the only common drawing aid that is supported in most popular software packages, including *Adobe GoLive* [Adoa], *Adobe Photoshop* [Adob], *Axon Solo* [Axo], *The GIMP* [GIM], *Macromedia Director* [Maca], *Macromedia Dreamweaver* [Macb], *Microsoft PowerPoint* [Mica] and *Microsoft Visio* [Micc].[7]

In an actual implementation a lot of issues remain to be solved. Some of the questions that arise are: How wide should the grid's base unit be? How could the interface support drawings that do not fit into the grid? Which point of an object is snapped to the grid, if the object does not span an integral number of base units? How do the layout and the grid fit after move, copy/paste or save and reload operations?

Some popular software has answers to these questions that turn out to be very helpful in practice. For example, in Microsoft Visio multiple base units with different snapping intensity are supported, effectively offering a grid with subintervals. Moreover, in Visio the base unit of the grid adapts to the current zoom factor, allowing for more detailed but still aligned drawing on a highly zoomed view. Importantly, all units of finer grids are always integral fractions of the units of the coarser grids, thus supporting consistent alignment all through out. Some of the other questions don't arise in Visio, because the sizes and positions of all objects are at all times aligned with some grid unit.

[7] All company and product names used in this text are trademarks or registered trademarks of their respective owners.

As another example, Macromedia Director allows objects to have a size and position independently from the grid unit. When snap-to-grid is active and an object is moved by drag-and-drop, the location where the mouse cursor grabs the object determines the part of the object that is aligned with the grid. This correlation is made explicit by visual feedback.

As we have seen in section 2.1.5 (item B), the Microsoft PowerPoint implementation of a regular grid as an alignment tool is seriously flawed.

Guidelines. Magnetic guidelines are another popular tool that often works well when a layout requires horizontal and vertical alignment but does not fit into the regular structure of a grid. Guidelines behave like gridlines, except that they are explicitly placed by the user at arbitrary positions. Again, the implementation of guidelines in Microsoft PowerPoint is unnecessarily limited.

Align and Distribute. In most popular software for interactive layout, including Microsoft PowerPoint, the frequent alignment operations are available as toolbar buttons. These typically cover the alignment of the top, bottom, left, or right border of the selected objects, where the object with topmost top border (or bottom, left, right, respectively) usually is immutable and the other objects are aligned. Also, middle (vertical) or center (horizontal) alignment is supported. The alignment operations mentioned always apply to only one dimension (horizontal or vertical) at a time. For alignment to be available, at least two objects must be selected.

A related operation is distribute, which takes three or more objects and distributes the space between them evenly while the two outmost objects are immutable. Again, this operation works only horizontally *or* vertically at a time.

3.2 Computer Supported Layout

Substantial work has been done in computer supported drawing and layout, the roots of this field of research dating back as far as to Sutherland's *Sketchpad* [Sut63]. All of these attempts have a tradeoff in common between the complexity of the input specification and the expressiveness of the system in terms of flexibility and accuracy of output. Interactive drawing aids generally use simpler input (namely, direct manipulation) than entirely automated systems. The latter often aim at reusing the same layout specification (usually provided in textual code) for multiple sets of content and to reuse the same content in different environments (e. g., different screen resolutions).

3.2.1 Interactive Drawing Aids

There has been some research on computer support for precise freehand drawing that overcomes the very restricted concept of a regular grid (cf. Sect. 3.1.2). Bier and Stone [BS86] propose a concept they call *snap-dragging*, which maintains the

idea of snapping the drawing caret to prominent locations. To determine geometrically prominent locations, snap-dragging uses the ruler-and-compass metaphor traditionally used by draftsmen. Based on explicit hints given by the user and some heuristics about typical editing behavior, snap-dragging constructs transient alignment objects – points, lines and circles – on the fly. A gravity function drags the caret to align with one or more of these objects, enabling fast and accurate drawing with little time spent in moving the tools and setting up the construction.

Bier and Stone's snap-dragging forgets about the relationships between shapes immediately after a shape is constructed, whereas Gleicher [Gle92] takes snap-dragging one step further and presents *augmented snapping*. Roughly spoken, in Gleicher's prototype system *Briar* the snap-dragging technique is used to specify constraints that are maintained from then on. This way, geometric relationships between drawing objects are invariant against editing operations like drag-and-drop. The specification of constraints through direct manipulation implies important simplifications compared to traditional constraint-based systems: Because constraints start out satisfied, there are no constraint-satisfaction problems to solve or state jumps to explain. There is no concern about conflicting or unsatisfiable constraints, since there exists at least one configuration which meets the constraints.

An essential feature of snap-dragging as well as augmented snapping is the immediate on-screen feedback to user actions and intentions. In snap-dragging, alignment objects are visualized such that they can easily be distinguished from concrete objects. In augmented snapping, the effect of applying a drag-and-drop operation while existing constraints are being maintained, is continuously displayed during the dragging action.

Other systems have been realized that support quick and geometrically precise drawing without using any grid or gravity function and without the explicit notion of constraints. Notably, Igarashi *et al.* have demonstrated various work in this field. *Interactive beautification* [IMKT97] infers relations between new strokes and existing line segments and thus generates geometrically accurate 2D shapes from plain and generally imprecise drawing input. In contrast to snap-dragging, interactive beautification "beautifies" an irregular input stroke after it has been drawn. Relations between the new stroke and existing objects are not explicitly shown. In *Teddy* [IMT99], on the other hand, two-dimensional drawing gestures are "inflated" into 3D models. Because the user interface is entirely based on direct manipulation gestures, no traditional UI widgets – buttons, sliders, menus, dialogs – are required for 3D construction using this technique.

Support for imprecise mouse movement and maintenance of spatial relations between on-screen objects is not restricted to the domain of freehand drawing. In [BNB00], Badros *et al.* present *SCWM*, an "intelligent constraint-enabled window manager". SCWM is a window manager for the X window system that handles relations between windows and enforces them when windows are moved or resized. SCWM implements a UI for constraint specification and visualization of active constraints. Constraints are implicitly inferred by augmented snapping and explicitly

specified using on-screen buttons. Both, buttons and constraint visualizations, use iconic representations of the supported constraints.

3.2.2 Automated Layout

As opposed to interactive drawing aids, by *automated layout* I mean a "specify-then-solve" approach. In such systems, the user describes the model by declaring relationships that must hold true and the system configures the model to meet these requirements. This approach allows a user to specify the important aspects of the design and have the system resolve the details.

A nested tabular layout can be seen as a set of spatial constraints. Typical UI toolkits and layout managers, widely deployed examples being Sun JFC/Swing [Sun02] and Microsoft Foundation Classes [Mic97], are based on this idea. These systems for development and presentation of window based user interfaces are designed to present the same UI widgets on different screen resolutions. Sizes and positions of the widgets are chosen at runtime, based on a simple built-in layout policy and a set of parameters specified by the programmer. Typical layout policies include row-major vs column-major, regular grid, or border layout, that can be combined in a hierarchical manner. Parameters specify details like minimum and maximum sizes. Hence, the layout as such is not automated; rather, the layout that was specified by the programmer is instantiated with respect to the conditions (screen resolution, window size, etc.) that prevail at runtime.

While the popular UI toolkits are merely based on *spatial constraints*, for more general applications it is favorable to be able to specify *abstract constraints*, too. Techniques that use markup tags, such as LaTeX [LaT] or HTML [W3Cb], present simple examples for spatial (e. g., *vspace*{*lineheight*}) and abstract (e. g., *level-1-heading*) constraints. A frequently used vehicle to achieve a rectilinear layout in LaTeX or HTML are tables. Tables in markup languages have similar characteristics as a grid: Table cells follow a strictly horizontal and vertical layout, but in contrast to a regular grid, the table's row heights and column widths are in general irregular. Cell sizes are dynamically adapted to meet the requirements of the contained objects on the one hand and the window or paper size on the other hand.

Cascading style sheets (CSS) are an extension to HTML that allows clean separation of structural markup and actual appearance. Badros *et al.* noted in [BBMS99] that the procedural specification of CSS 2.0 [W3Ca] is complex and sometimes vague, in any case not easy to understand and implement. They propose a constraint-based extension to CSS 2.0, which is backwards compatible and has the advantage to declaratively specify the standard. Badros *et al.* argue that a constraint-based approach to style sheets would greatly simplify the implementation of rendering engines as well as the web designers' use of cascading style sheets. Moreover, they suggest some more general and flexible style specification syntax that can easily be implemented using constraints, but is not easy to implement using the procedural approach of current CSS.

The simplicity of the markup approach is reflected by the fact that the order in which objects are specified as input directly determines the order in which formatted objects appear in the presentation. In contrast, general constraint-based multimedia tools take as input a constraint network that is not simply unidirectional. Prominent examples for research systems that process multidimensional relations between layout objects are Feiner's *GRIDS* [Fei88], Weitzman and Wittenburg's system [WW94] and Graf's *LayLab* [Gra95].

GRIDS is a rule-based system that uses rectilinear layout to enforce a consistent look in multi-page documents.

Weitzman and Wittenburg's system parses relational grammars that describe a layout independently from the content. Once the content is available, the grammar is applied, establishing constraints between concrete objects. Thus, the grammar can be reused to lay out different content and the same content can be rendered differently for another purpose by using another grammar.

Graf's LayLab provides an architecture that is built around a *layout constraint engine*, thus enabling different execution modes such as fully-automated vs. semi-automated fashion, editing with automatic beautification, and interaction.

3.2.3 Ancestors of the Smart Grid

The *smart grid* is a new approach to computer supported layout for business slides, which is discussed in detail in section 4.1. In the following passage the smart grid idea is motivated in the light of other research.

Remarkably, although the constraint-based approach to automated layout is very intuitive and configurable for a wide range of applications, to date it is not implemented by any widely used product. Lok and Feiner [LF01] suggest two reasons for this observation. First, sophisticated constraint-based systems inherently require the user to employ complex representations of the relationships between content objects. This poses formidable user interface problems that must be solved before actual users can productively use the system. Second, because an automated system has a much smaller space of parameters to work with than a human designer, good graphic designers will consistently outperform automated systems in terms of quality of the result. Reportedly, the domain of yellow page pagination is an exception to this observation: Johari *et al.* [JMPS97] have applied a weak stochastic optimization algorithm to a sample of real telephone-directory data and produced a layout that was "better" (by a range of measures) than the published one.

The conclusions I draw are as follows: The reason to deploy (semi-)automated layout tools comes from the desire to achieve a constant, reliable quality of the output especially for routine tasks, and at the same time is an attempt to improve the efficiency of human labor. The most promising application domains from a pragmatic point of view are those with little complexity, that can be sufficiently described with only a small range of different constraints. The domain of business slides fits into this picture: Layout is complex enough to consume substantial resources of manpower, but it is simple enough to benefit from a constraint-based layout support, such as offered by the smart grid approach.

The smart grid is a hybrid of the regular grid and the popular guideline tool, which were both discussed in section 3.1.2. The basic idea is to implicitly generate guidelines when needed and use them subsequently to align shapes. In contrast to the guidelines generally found in drawing applications and notably in Microsoft PowerPoint, which are static except for explicit modifications by the user, the smart grid is dynamically updated. In this regard, the smart grid resembles a web page that is laid out using HTML tables: Objects are arranged in an irregular grid, that is stretched and bent as required to accommodate the (preferred) sizes of the contained objects as well as the available real estate on a page or screen.

The smart grid integrates guidelines with constraints. This approach exhibits some resemblance to augmented snapping [Gle92]: Drawing objects snap to alignment objects, generating constraints that are enforced during later editing operations. While maintaining the basic idea, in my approach the variety of alignment objects is restricted to horizontal and vertical lines. This appears to satisfy most alignment requirements that arise in the context of business slides and makes the necessary computation for constraint solving a lot easier.

It is important to note another fundamental difference: While in augmented snapping, any two objects on a page may or may not be related by some constraint, in the approach presented in this work all objects on a slide are always related through the smart grid.

3.3 State of the Art – Résumé

The review of literature shows that a lot of research has been done in the fields of menu based user interfaces, as well as layout automation. Nonetheless, Microsoft PowerPoint does not employ any of the techniques suggested by research, neither constraint-based layout support nor any of the menu techniques that have shown to be more efficient than standard linear menus. On the other hand, none of the discussed research systems is designed for business slide creation, and obviously there is also no standard software available that is better suited to this task than PowerPoint. (If there was, one can assume that professionals would use it, but everybody uses PowerPoint.)

The smart grid approach and the user interface concept that are presented in the remainder of this work have the goal to remedy this situation: I developed a working prototype that implements a new and particularly efficient user interface for constraint specification. Based on the think-cell software, my prototype is designed as an add-in to extend PowerPoint with constraint-based layout capability.

4 Interaction Concept

This chapter deals with the new *smart grid* approach. First, the concept of the smart grid is explained. Then, the meaning of the term *usability* will be examined and available techniques for user interface design under Microsoft Windows will be discussed. Finally, a user interface for efficient slide layout will be developed.

4.1 A New Approach to Slide Layout

Computer supported layout as it is seen in a range of academic projects to date has not made it into any widely used applications (Chap. 3). Microsoft PowerPoint, on the contrary, has a huge user population and is used for a variety of different purposes in business, academic and even private life. But layout support in PowerPoint is limited to some macro-like tools that move shapes based on absolute positions and temporarily established relations (Chap. 2).

4.1.1 Plain PowerPoint is Not Enough

A closer look at the output that is created with Microsoft PowerPoint reveals that there is a lot more to an informative and pleasing slide layout than just fixed pixel positions. Figure 9 shows a typical slide as it is used in business consultancies. Some basic layout rules – including symmetry, regular spacing between shapes, minimal text wrapping and consistent font sizes – are reflected by this example. Studying more slides helps to see more rules of this kind and to understand how they interact.

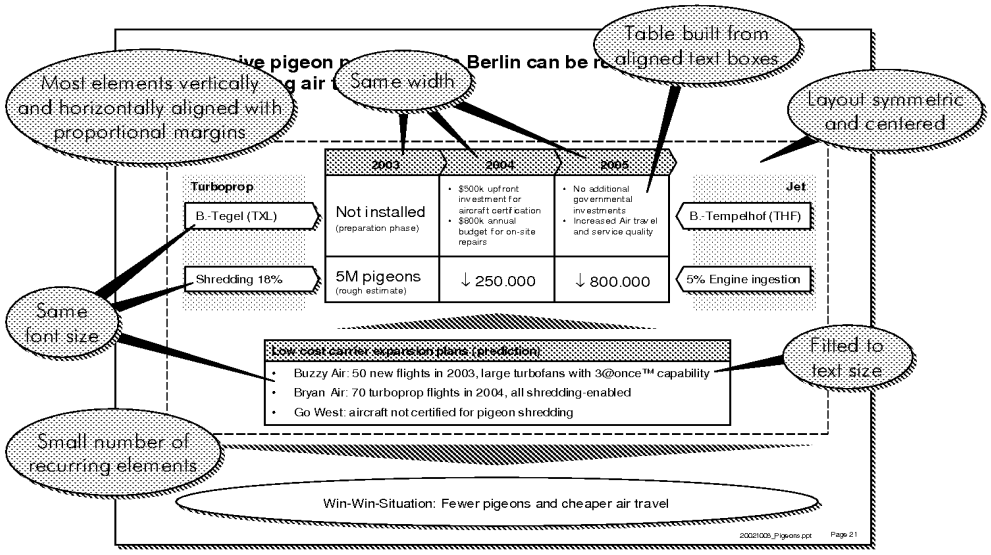


Figure 9: A real life example of a consultancy slide, highlighting some basic rules that make a pleasing over-all layout. Based on expert ratings, the layout of this slide takes about 15 minutes in plain PowerPoint, not taking into account the time for text entry. (slide text replaced for confidentiality reasons)

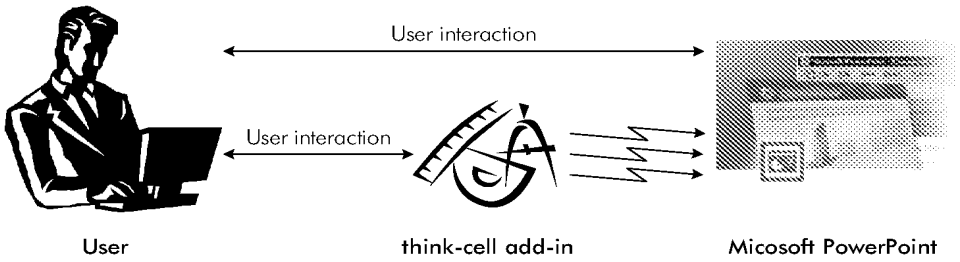


Figure 10: The user's view: The think-cell add-in helps to quickly generate satisfying results in Microsoft PowerPoint. Besides the specific UI widgets, all PowerPoint features behave as usual.

With these observations in mind, the extension of PowerPoint with some automated layout optimization technique appears promising. The user should be able to specify intentions, which are recognized by the program as numeric constraints. Constraints can be solved and the intended layout can be arranged by the computer. Moreover, when part of the composition is changed by the user, the over-all layout can be updated to preserve the constraints that were specified earlier.

The *think-cell layout* add-in for Microsoft PowerPoint provides these features. From the user's point of view, the think-cell software adds some new UI widgets to the Microsoft PowerPoint user interface (Fig. 10), while the rest of PowerPoint looks and behaves just the way it always did.

The challenge of this approach is to model the characteristics of optimal layout in a way that can be computationally solved. This leads to the idea of a "smart grid": Constraints that can be expressed by a grid-like structure are almost sufficient to describe a layout model for business slides.

4.1.2 The Smart Grid Concept

The smart grid concept provides an abstraction from exact pixel positions and absolute sizes. The additional abstraction layer simplifies user input, requiring greater machine work to interpret the input and to translate the user's incomplete layout specification into a fully specified slide layout. The gap between the abstract, incomplete layout specification and the final layout is bridged by generic constraints that formalize design rules for esthetic layout.

Figure 11 provides an overview of the automation process. This work focuses on the user interaction, whereas the solving of numeric constraints that is required to implement a self-adjusting smart grid is beyond the scope of my thesis.

Terminology

The *smart grid* is a dual concept based on *smart gridlines* and *smart elements*. Figure 12 visualizes how the components of the concept interact. Smart elements are bound to horizontal and vertical gridlines; elements that are bound to the same gridline are effectively aligned.

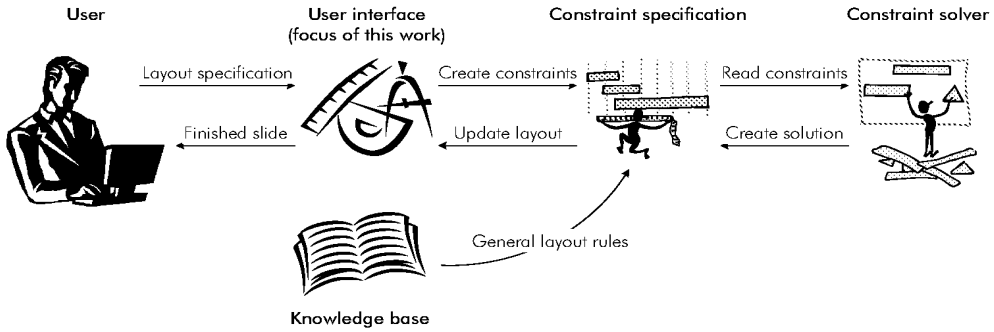
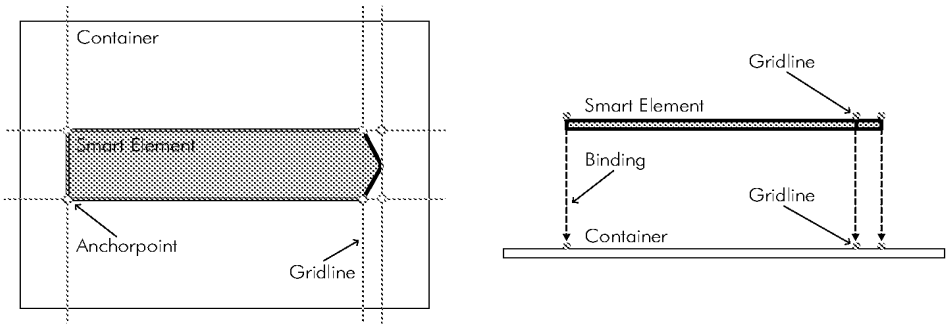


Figure 11: The focus of this work in the context of the think-cell application

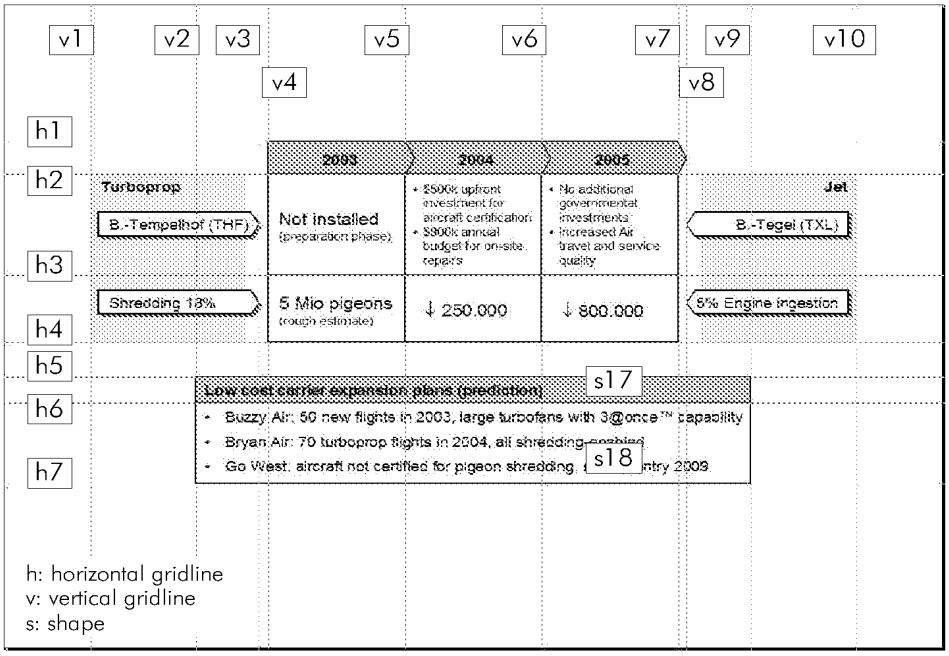


(a) The user's view ("from above") (b) The programmer's view ("side view")

Figure 12: Schematic interaction between smart elements and smart gridlines

The notion of the smart grid refers to the entirety of horizontal and vertical gridlines plus certain constraints induced by the contained elements plus constraints explicitly specified by the user. The smart grid instantiates a system of numeric inequalities that represent the *constraint specification* (an example is provided in Fig. 13). The system of numeric inequalities is taken as input by the *constraint solver* subsystem. Constraints are weighted and thus the inequality system can always be solved, to a more or less satisfying degree. The solution is then imposed on the layout by moving the gridlines; any shapes bound to those gridlines are implicitly moved along with them. When gridlines are moved, sizes and positions of the bound shapes may change, but spatial relations and alignments are implicitly maintained.

Smart elements replace the templates that are traditionally used to quickly set up a slide. The complexity of the templates and the smart elements, respectively, ranges from simple arrows of a certain shade to difficult arrangements such as Gantt charts. Ordinary templates simply consist of a prearranged group of PowerPoint shapes that is very fragile and easily distorts when it is adapted for a specific slide. On the contrary, smart elements serve the same purpose but have a hard-coded internal structure. Smart elements cannot be torn apart by mistake. They also actively rearrange their internal layout according to the conditions of their requirements, such as size, proportions and text length.



(a) Shapes and gridlines as seen by the user (life screenshot from my prototype with additional annotations)

```
                ...
        v5 - v4 = v6 - v5
        v5 - v4 = v7 - v6
        v4 - v3 = v8 - v7
        v3 - v1 = v10 - v8
                ...
        v9 - v2 ≥ minWidth(s17)
        v9 - v2 ≥ minWidth(s18)
             v4 ≥ v2
             v9 ≥ v7
                  minimize( v9 - v2 )
                ...
        v4 - v3 > minGap
        v8 - v7 > minGap
min(v) - leftBorder = rightBorder - max(v)
                ...
        h5 - h4 > minGap
        h2 - h1 = fixed
 min(h) - topBorder ≥ titleHeight
                ...
```

(b) A system of constraints as it is sent to the solver

Figure 13: The layout of a slide can be determined by solving a constraint system for relations between gridlines.

Each smart element exports those of its properties which have impact on the over-all slide layout. This includes measures such as minimal and preferred sizes, and gridlines of internal shapes that can be used for alignment. At the minimum, the gridlines of a smart element's bounding box are always exported. A generic wrapper creates the required smart element interface for native PowerPoint shapes.

Since the edges of an element's bounding box are always bound to some gridline, the element's corners always lie on the intersection of two gridlines. I call the corners of a smart element *anchorpoints*, because the four anchorpoints "anchor" the smart element's position within the smart grid. The resulting grid of horizontal and vertical gridlines is not necessarily regularly spaced, but each smart element always spans an integral number of grid cells and smart elements may overlap each others arbitrarily.

The slide is laid out in a hierarchical fashion, with a transparent *container* that spans the entire slide as the root. Containers are smart elements themselves. When a smart element is inserted into a container, the smart element's gridlines are bound to gridlines of the container. If there are already container gridlines available at the required positions, the smart element's gridlines are bound to them, implicitly creating a sustainable alignment relationship with any other smart elements bound to the same gridline. If at the required position there is not yet a container gridline available, a new gridline is created on the fly. Thus, all gridlines from all smart elements propagate through the entire hierarchy and are ultimately bound to some gridline of the root container. On the other hand, when a container gridline is left with no smart elements bound to it, it is implicitly removed.

Only the smart grid of the root container is visible and accessible to the user. Since a root container spans an entire slide and has all elements of the slide bound to its gridlines, the optimal solution to the root container's smart grid provides an optimal solution to the over-all slide layout. Smart grid and container have a one-to-one relationship and each slide has its own distinguished root container. Thus, each slide refers to exactly one smart grid entity for layout optimization and the solution for a constraint specification as represented by a root container's smart grid is applicable to a single specific slide.

Constraints Reflected by Grid Structure

Technically speaking, the smart grid holds a collection of arbitrary constraints that must be satisfied by a layout to resemble the user's intention. Practically speaking, a large fraction of the constraints required to specify a typical layout can be expressed by means of horizontal and vertical gridlines; it is this observation that makes the "smart grid" approach so appealing.

The concept of gridlines that are bound to other gridlines is used to specify alignment relations between two or more smart elements. For example, a set of such relations imposes a tabular layout with shapes in the same row having the same height and shapes in the same column having the same width. Moreover, if we take the total ordering of gridlines in each dimension as yet another constraint,

the 2-dimensional total ordering of the smart elements is also specified by the smart grid. Finally, gridlines serve to represent the constraints that are exported by the smart elements bound to them, in particular preferred and minimum sizes.

Two minor modifications are required to tailor this interpretation of the smart grid for practical use: Firstly, white space between elements plays an important role in the visual appearance of a slide and must be preserved. Therefore, some smart elements have a built-in margin surrounding them. Furthermore, I use an explicit *gap* smart element which can be inserted into the smart grid to request an additional amount of white space. Secondly, the total ordering of gridlines must be weakened, because a total ordering of shapes is not always intended. For example, shapes that reside in independent areas of a slide may well be moved and sized independently, if this leads to a better layout.

Further constraints that help to abstractly specify layout intentions can be represented by the smart grid. For instance, the user can *pin a gridline* to a certain position, meaning that the solver must leave those gridline's position untouched. Another option is to select a group of gridlines and require equable spacing between them.

Constraints That Cannot Be Expressed by a Grid

An analysis of a large number of slides from the target application domain showed that grid-oriented constraints do not suffice to express all frequently used layout intentions. In order to make the system applicable for a reasonably large set of use cases, support must be provided for certain constraints between elements that are not necessarily related through the grid. This includes constant proportions between the sizes of two elements – 1:1 (same size) in most cases –, as well as same style, same color, same font size and the like.

Constraints Spanning Multiple Slides

In contrast to other grid-based layout systems (e. g., Feiner's GRIDS [Fei88]), the smart grid is not fix but dynamically changes to reflect the requirements of the contained shapes. Unlike the guidelines in plain Microsoft PowerPoint, which have global scope, the smart grid is primarily a tool for local layout of a single slide. The concept, however, could easily be extended with inter-slide constraints and thus the smart grid could also support a uniform layout across a set of slides or an entire presentation.

A priori, gridlines in the smart grid have a scope local to the current slide. To support some kind of master layout that applies uniformly to each slide, it suffices to introduce the concept of *global gridlines*: Gridlines that appear at the same position on each slide. Positions of global gridlines cannot be optimized by local constraint solving; therefore, to the solver global gridlines appear just as pinned gridlines. As of now, global gridlines are not yet part of the prototype implementation.

Another user request that was revealed by the field study (Chap. 2) cannot easily be integrated into the smart grid concept: So-called *stickers* are visual elements that appear on a couple of slides throughout a presentation, always in the same size, position and style. In Solo [Axo] there is a feature called *named objects* that explicitly supports this concept. It will be up to future work to represent the sticker concept with the smart grid approach.

4.1.3 Implementing Esthetics by Rules

The argument for a substantial performance gain by computer supported layout says, first, that it is easier and faster for the user to specify abstract layout intentions as opposed to concrete pixel positions. Second, the argument assumes that the translation from an abstract specification to a concrete layout can be done by an automated process and third, that on a recent personal computer this process runs fast enough to support the user interactively.

An abstract layout specification allows for a much faster input, because it carries much less information than an exact, pixel-based specification. As a consequence, an abstract specification typically is an underspecification of the desired outcome. Just as in inter-human communication, the pure user input leaves room for ambiguities that must be resolved. In inter-human communication, ambiguities are resolved by supplementary input from other media, namely gestures and tone, and by *conventions*. Conventions go a long way to resolve ambiguities in the literal information, because by convention we have certain preconceptions about the meaning of a piece of information depending on the situation, the relationship between the speaker and the listener, and so forth.

Similarly, in computer supported layout conventions help to resolve ambiguities in the user input. More technically speaking, the constraints implicitly given by the smart grid and the constraints explicitly specified by the user are supplemented with a collection of generic constraints that comprise conventions – or rules – about esthetic layout and corporate design (see *knowledge base* in Fig. 11). Those generic constraints bridge the gap between the abstract and underspecifying user input and a concrete layout.

The actual definition of generic constraints must come from expert knowledge as well as from studying a substantial set of sample slides. The constraints must be weighted and the formulation of the constraints as well as their weights must be carefully tuned to the constraint solving algorithm. Those details are beyond the scope of my thesis, but to fortify the feasibility of the smart grid approach and to give an impression of what conventions in computer supported layout may look like, I propose some rules for this purpose. This listing of rules is certainly incomplete, inaccurate and open to discussion.

- A slide consists of static objects such as headline and page number, and a set of dynamic objects, the body.

- The body of a slide is centered between the static slide margins, which are part of the master layout.
- Spacing between elements in the body is uniformly distributed; when there are only few elements on a slide, spacing is larger than when there are a lot of elements.
- Elements that contain text or data graphs have a preferred aspect ratio of 4:3.
- Text wrapping should be avoided, if possible; any wrapped line of text should at least contain three words. Line wrapping should avoid syllabication. If necessary, long words may be wrapped but correct syllabication must be respected (in contrast to what plain PowerPoint does).
- A single line of text should be centered in its containing element; multi-line text should always be left-aligned.
- A slide should contain as few different font styles as possible, preferably no more than three different font sizes plus bold and italic. Static elements like headings and page numbers must have the same font size and style throughout an entire presentation.
- Font sizes should default to 14 pt. For large chunks of text that must be wrapped into multiple lines, text size may be decreased to 12 pt or even 10 pt. No text must ever be smaller than 10 pt. Small chunks of emphasized text may be as big as 18 pt.

4.1.4 Review of the User Feedback

The smart grid approach is strongly supported by the qualitative results from the user study (Sect. 2.1). Figure 14 shows an excerpt from the user ratings of the deficits in the Microsoft PowerPoint user interface. Assuming that the smart grid automatically and instantly applies adequate proportions and perfectly aligns all shapes on a slide, a large fraction of the reported problems would be solved or become irrelevant:

- The **regular grid** (items[8] B1 and B2) and the **guidelines** (item B3) are completely replaced by the smart grid. Smart gridlines emerge whenever a shape is placed. Shapes and smart gridlines can be snapped to other smart gridlines. Smart gridlines move autonomously to adjust the over-all layout. Alternatively, they can also be moved manually just as usual guidelines. Visibility of smart gridlines is improved over the hardly visible PowerPoint grid (item B4).
- Any tasks related to **tabular layout** (item D) are easily solved with the smart grid, because the smart grid itself is simply a generalization of a tabular layout with rows and columns.

[8] Item numbers refer to the detailed description of Microsoft PowerPoint deficits in section 2.1.5.

	User 1 (Beg.*)	User 2 (Adv.*)	User 3 (Adv.*)	User 4 (Exp.*)	User 5 (Exp.*)
1	sticker	grid	grid	table support	bulleted lists
2	pics	text wrapping	pics	navigation on zoom	masterslide
3	masterslide	bulleted lists	bulleted lists	grid	sticker
4	grid	navigation on zoom	booz balls	sticker	grid
5	bulleted lists	pics	keyboard support	bulleted lists	keyboard support
6	resizing shapes	booz balls	resizing shapes	keyboard support	table support
7	navigation on zoom	keyboard support	inadvertent selection	pyramid charts	b/w hatching
8	text wrapping	inadvertent selection	masterslide	text wrapping	
9	inadvertent selectiony	sticker	guidelines	inadvertent selection	
10		b/w hatching		pics	
11		resizing shapes		guidelines	

* Beg.: Beginner; Adv.: Advanced; Exp.: Expert

Figure 14: Significance rating of PowerPoint deficits, highlighting all issues that are potentially remedied by the "smart grid" approach

- Precise **resizing of shapes** (items B5, B6 and B7) is a typical example for a time consuming exercise that easily propagates throughout an entire slide. With the smart grid holding all shapes in place and aligning them, the need to manually resize drawing elements is eliminated.
- Microsoft PowerPoint has no support other than the scroll bars to **navigate a slide on high zoom factors** (items B5 and E2). However, when the smart grid guarantees perfect alignment, the need to view a slide with high zoom factors is drastically reduced and navigating the slide becomes much a smaller issue.
- Automatic **text wrapping** (item A3) must be solved to facilitate a self-adjusting layout. This is a standard problem which can be solved by a canned wrapping and syllabication algorithm.

4.2 Interaction Toolbox

Before we can define a user interface for a certain purpose, we must know the user interface techniques that are available, and their specific advantages and disadvantages. Let's start with a definition of usability metrics and design objectives, and then discuss some selected aspects of user interfaces for Microsoft Windows applications. My terminology for user interface widgets and interaction techniques is based on The Microsoft User Experience [Micb].

4.2.1 Usability Metrics and Design Objectives

In an attempt to understand the very general concept of usability, I suggest the definition from ISO 9241 as found in [DFAB98]:

> *"By* usability *we mean the* effectiveness, efficiency *and* satisfaction *with which specified users achieve specified goals in particular environments."*

Effectiveness. "The accuracy and completeness with which specified users can achieve specified goals in particular environments."

For example, a fork is not effective to eat soup with. Effectiveness comes before efficiency: If a goal cannot be achieved at all, it can certainly not be achieved with any reasonable amount of resources. On the other hand, although I have doubts about its efficiency, Microsoft PowerPoint is definitely effective for the creation of slides. This is proven by the sheer amount of PowerPoint presentations that have been created to date.

Efficiency. "The resources expended in relation to the accuracy and completeness of goals achieved."

For example, I claim that PowerPoint is inefficient if perfectly aligned layout is the goal, because the pixel-oriented layout specification can be either fast or accurate, but hardly both at the same time.

Satisfaction. "The comfort and acceptability of the work system to its users and other people affected by its use."

It is certainly desirable that the use of software does not adversely affect people's health and it is also proven that constant dissatisfaction has negative impact on productivity and creativity. Therefore, it's the developers' job to create software that is not only highly effective and efficient, but also makes the user feel in good hands and pleases her from an aesthetic point of view.

Effectiveness, efficiency and satisfaction set the goals for a usable interface; however, these terms do not say anything about how the goals can be reached. There are a myriad of terms, methods and rules that describe the characteristics of "good usability". The following design objectives subsume some of the most important aspects of usable software (again, quoted from [DFAB98]).

Learnability. "The ease with which new users can begin effective interaction and achieve maximal performance."

For the present work, this objective requests that skilled PowerPoint users should be able to use the smart grid with little training or instruction. To achieve this, all the functionality already known from plain PowerPoint must behave exactly the same with or without the add-in. The behavior of any additional UI components should be modelled after the example of the existing PowerPoint user interface, but it might be helpful to make new features visually distinctive from the original PowerPoint, so that the user can easily tell

which widgets relate to the abstract smart grid paradigm and which provide traditional pixel-oriented functionality.

Flexibility. "The multiplicity of ways the user and system exchange information."

Generally speaking, a system should not restrict the user to one singular interaction pattern in order to achieve a certain goal. There should be alternate ways to interact with the system, because different users and different situations have different demands. For example, when an expert user is operating with both hands, keyboard shortcuts may provide a very fast way to interact with the system. For a novice as well as for somebody who operates a telephone with the other hand, it is probably helpful when the same result can be achieved with the mouse instead of the keyboard.

Robustness. "The level of support provided to the user in determining successful achievement and assessment of goals."

In my case, this objective requests that the system must appropriately react to any possible constraint specification. In all reasonable cases, a solution should be found that meets the user's expectations or at least does not completely distort the layout. As another consequence of this objective, the system should behave responsive and manageable in all states and all changes to the layout must be reversible. Full support for multiple undo helps to minimize the costs for recovering from erroneous input. This not only conforms to the objective of robustness, but also encourages explorative user behavior and thus improves the learnability of the system.

4.2.2 Keyboard-based Interaction

As of today, the keyboard is the only practical means for text and data entry, although there are alternative input devices for specific situations away from the office desk, or for the disabled. In general, for expert users the keyboard allows very fast input of text as well as of commands. For novice users, however, maneuvering a complex application by keyboard is usually hard because the assignment of commands or shortcut keys to functionality is arbitrary and therefore requires constant look-up or upfront learning.

Especially in graphical context as in layout, the keyboard is of limited use because keys poorly map to spatial movements or other non-text and non-linear activity. In a pixel-oriented application such as plain PowerPoint, the cursor keys have some value for very fine-grained and precise movements, which is hard to achieve with the mouse. With an abstract approach to layout, precision at the level of pixels is realized by the solver; constraint specification is spatially coarse-grained and therefore the role of the keyboard is merely that of an emergency exit for situations when the mouse cannot be used.

In general, the user interface should be designed in a way that almost any action can be performed using either the keyboard or the mouse. This way not only a range

of different needs is served, but also the need to switch frequently between mouse and keyboard is reduced. This is especially important for optimal performance of expert users because the frequent relocation of the hands from mouse to keyboard and back requires additional attention.

Modifier Keys. Usual modifier keys are SHIFT, CTRL and ALT. When held down while another key is pressed, the modifier keys modify the function of that other key. Modifier keys are used with alphanumeric keys and function keys as well as sometimes with mouse interaction. Although it often appears convenient to assign multiple – usually similar – functions to the same key with different modifier keys, the multiplicity of key combinations usually does not allow single handed input and also may confuse users.

Shortcut Keys. There is a number of common shortcut keys – actually, key combinations – that should be supported by all Microsoft Windows applications (refer to [Micb] for a complete list). In particular, those common shortcut keys should not take on other functions than the standardized ones. This idea supports the principles of consistency, familiarity and predictability and thus enhances the learnability of the system.

Function Keys. To the function keys F1 through F12, similar considerations apply as to shortcut keys. Moreover, while shortcut keys usually have at least one character as a reminder of their function – e. g., CTRL-C for copy – the assignment of features to function keys appears entirely arbitrary and is therefore especially hard to learn. In addition, function keys are usually located along the upper brim of the keyboard and often are especially small on laptop computers, meaning that additional effort and attention is required to reach them. For these reasons, the use of function keys for application specific features is generally discouraged, although the default functions that are suggested by [Micb] should be implemented.

4.2.3 Mouse-based Interaction

The mouse is a standard input device for personal computers just as much as the keyboard. While the keyboard is ideal for discrete input, the mouse is convenient to interact with a (pseudo) continuum in 2-dimensional space. The most commonly used mouse interaction patterns are *point-and-click* and *drag-and-drop*. The *mouse-over* pattern, that occurs when the mouse is moved and then rests with no buttons pressed, is often used to support learnability: An on-screen control at the current mouse pointer position highlights and sometimes a short explanation of the control's function is given by a tooltip.

In Microsoft Windows, there are always at least two mouse buttons that can be used in the user interface. Some mice have additional buttons and a wheel, that can be used to provide shortcuts to frequently used functions. However, it must remain possible to access an application's full functionality with only two

buttons. Mouse input can be combined with modifier keys to serve more functions, but when modifier keys must be hold down, the mouse can no longer be used single-handed. For notebook users, the requirement to hold down a modifier key and a touchpad button and move the finger on the touchpad at the same time, is especially cumbersome.

Commands and Menus

The concept of a menu emerged as an answer to hard-to-learn and error prone command line interfaces. A menu visually presents the set of available commands in a hierarchical structure. Although it is possible to navigate menus with the keyboard, they are especially easy to use with the mouse. The basic variant of a menu is a menu bar, which has a fixed location at the top of the screen or window, with submenus dropping down on request. For optimal learnability and high performance use, menu items should always show up in the same absolute position. Therefore, temporarily not available menu items are disabled rather than removed, thus leaving the over-all arrangement of menu items intact.

Context Menu. A context menu, also referred to as a *shortcut menu* or *pop-up menu*, is not always visible but appears in place at the mouse pointer position. By convention, the context menu is triggered with a right mouse button click. It offers selected functions that apply to previously or implicitly selected objects. For the smart grid, context menus provide a means to add user interface widgets on top of PowerPoint without adding clutter to all-time visible menus and toolbars.

Circular Context Menu. Although menus are traditionally arranged in a list style, there are good arguments for a circular arrangement of menu items, also referred to as a *pie menu*. Obviously, a concentric layout implies that the way for the mouse cursor to move is the same for each item and almost minimal. Moreover, research suggests that in a circular layout with a wedge for each menu item, items are easier to remember and easier to hit, than in a list-style menu. Selected research on this topic has been presented in section 3.1.1.

There are some drawbacks of a pie menu layout. For example, a pie menu can only hold a comparatively small number of items. Slicing a circle into 8 wedges appears somehow natural and matches well to the observation of "7 ± 2 chunks of information" that human memory can handle in one operation (e. g., [Bad97]), but list-style menus in PowerPoint contain twice as much items. Furthermore, hierarchical structure does not as easily map to circular layout as to list-style layout. Although there has been some promising research with multi-level pie menus called *marking menus* [TK95], it seems reasonable for a real-world application to restrict pie menus to only one level when they are first introduced to a non-academic user population.

I decided to try and use pie menus for the smart grid, not only because of their greater usability but also to visually distinguish the smart grid specific features from

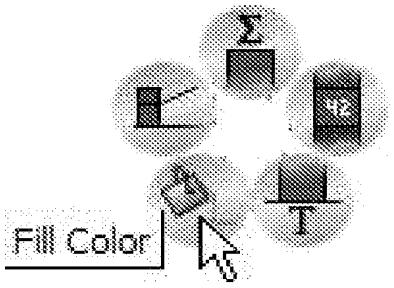


Figure 15: A circular context menu for bar chart settings as it appears in my software, with an explanatory tooltip

plain PowerPoint and to convey an innovative look-and-feel. However, care must be taken to leverage the advantages of pie menus without introducing new usability problems. For example, menu items must not obscure the visual context, which is a contradiction to readable and telling text labels. I solved this contradiction by using circular buttons that cover only a relatively small area, with professionally designed icons and additional tooltips. While the visible buttons cover a relatively small area, the entire pie wedge remains responsive to mouse clicks thus maintaining the easy-to-hit advantage. Figure 15 shows a screenshot of a circular menu as it appears in my implementation.

Floating Menu. Floating menus are a variant of context menus that appear when the mouse pointer hovers over an object, without requiring an extra mouse button click. Microsoft has implemented a floating menu for images of a certain size in Internet Explorer Version 6. I decided against implicitly appearing menus because they are unnecessarily distracting when the user does not intend to use them.

Toolbar. Toolbars provide a virtual keyboard and use a button metaphor for user interaction. Unlike menus, toolbars usually have a flat, non-hierarchical layout. As opposed to shortcuts and function keys on the physical keyboard, toolbar buttons have the advantage that there are always as many of them, as required to control a certain application, and every toolbar button can have an icon and a descriptive tooltip text as a reminder of its function.

In current office applications, toolbars are a prime user interface component. In fact, toolbars are so convenient to design and to use, that those applications tend to be cluttered with toolbars. In an attempt to pack even more functions into a toolbar, recently software developers started to use some kind of hierarchical menu popping up. The difference of menus and toolbars thus boils down to menus being labelled with text and toolbars being labelled with icons.

In an attempt to avoid yet more toolbar clutter, I refrained from adding an additional toolbar as a user interface to the smart grid. I only offer a menu for new smart elements that appears next to the AutoShapes menu, which is well-known to PowerPoint users. Since smart elements behave similar to PowerPoint AutoShapes,

I hope that the placement of the smart-elements menu will help PowerPoint users to master the new functionality.

Mouse as Universal Design Tool

While using the mouse to navigate virtual controls provides the opportunity for very rich and flexible UI widgets, the mouse can also serve as a universal tool to manipulate the objects of interest. In a drawing application, for example, the mouse can behave as a pen, or a rubber, or an airbrush, and so forth. Similarly, the mouse can act as virtual hands, scissors and glue to create and modify a slide layout on screen. The term *direct manipulation* defines some important aspects of this kind of interaction, while *snapping* is a technique that adds spatial precision to otherwise free hand drawing tools.

Direct Manipulation. The mouse enables what is called direct manipulation in a graphical user interface. The Introduction to HCI [DFAB98] cites Ben Shneiderman with the following definition of a direct manipulation interface:

- Visibility of the objects of interest
- Incremental action at the interface with rapid feedback on all actions
- Reversibility of all actions, so that users are encouraged to explore without severe penalties
- Syntactic correctness of all actions, so that every user action is a legal operation
- Replacement of complex command languages with actions to manipulate directly the visible objects

With regard to computer supported slide layout, the principle of direct manipulation requests that the layout is interactively arranged in the 2-dimensional space, thus implicitly specifying numeric constraints, rather than typing in the numeric inequality system. In Microsoft Chart, as a contrary example, the direct manipulation principle is broken (Sect. 2.1.5, item C1).

Snapping with Gravity. Although it is an important achievement of the mouse that it enables seemingly continuous navigation in a 2-dimensional space, when precise mouse pointer movement is desired it often helps to prune the continuum and have the mouse pointer discretely snap to prominent positions. This kind of mouse behavior is typical for construction, drawing and – as in my case – layout applications.

There is a range of variations to this theme: Some applications have a strict grid and any input from the mouse is interpreted as if the mouse was precisely pointing to a crossing of gridlines. Such a grid implements an entirely discrete 2-dimensional space. Another approach lets the mouse move continuously for the most

part, but snaps to certain snapping objects when the mouse pointer is sufficiently close. Snapping objects may be virtual guidelines that serve the purpose of ruler and compass, or the mouse may also snap to drawing objects that are part of the document. A gravity function determines what "sufficiently close" means and resolves conflicts between two or more snapping objects which are about equally close to the mouse pointer location. There may be gravity functions of different strengths, relating to the significance of different snapping objects.

A lot of research has been conducted on snapping (see Sect. 3.2.1) and it turns out that the distinction between *mouse pointer* and *drawing caret* helps to understand and efficiently use snapping mechanisms. In this distinction, the mouse pointer always moves smoothly in the continuum, exactly reflecting the movements of the mouse, while the drawing caret indicates which position in the document will be referred to when a mouse button is pressed. It is the caret that is snapped to discrete positions while the pointer continues to move freely. Some recent software, including Microsoft Visio [Micc] and Autodesk AutoCAD [Aut], adheres precisely to this definition, while other programs like Microsoft PowerPoint do not explicitly display the caret but implicitly apply mouse interactions to the closest snapping position. In the latter case, it may be a good idea to highlight the snapping object when snapping occurs, so that the user has a chance to move the mouse further away if snapping is not intended.

As for zoomed views, the velocity of gravity functions is usually based on screen coordinates, not document coordinates. As a consequence, in an enlarged view drawing objects appear more detailed and the sphere of influence of a snapping object becomes smaller in relation to the drawing. Thus, in the case of many snapping positions being close to each other, using a highly zoomed view makes it easy to snap to the desired position.

4.2.4 Mode Switch

In the present scenario of one application being built on top of another, there is a radical method to avoid interference of distinct applications' user interfaces: I could implement some sort of escape key that switches between two completely independent user interfaces, one for each application. This is in fact how the Microsoft implementation of OLE (Object Linking and Embedding) appears to the user, a technique that is used to embed charts, diagrams, images, movies, formulae and other complex objects in documents created with PowerPoint, Excel, or Word. Double-clicking an OLE object switches to the UI context of some other application. The advantage is obvious: Menus, toolbars and shortcut keys are completely changed by the mode switch and can all be used without any restrictions that emerge from another application using the same UI widgets.

But this advantage is only of theoretical relevance. If some well-known PowerPoint shortcut key takes on a completely different meaning in "think-cell mode", then mistakes in the user input and user dissatisfaction are almost guaranteed. Another problem of this approach is the technical and cognitive overhead that is

induced by a mode switch. Since the think-cell add-in very closely interacts with traditional PowerPoint functionality, frequent mode switches would be required, slowing down the interaction and adding to the confusion on the user's part.

Therefore, I decided to blend in to the existing PowerPoint user interface in the least intrusive way possible. I leave the PowerPoint interface intact and restrict myself to UI components that are not yet used by PowerPoint itself.

4.2.5 General Interaction Patterns

In an interactive drawing or layout application, there are few goals that can be reached with one singular command as a button click. Most goals require the sequential combination of several interactions, which I call an *interaction pattern*. For example, to change an object's color, it is necessary to select the color feature as well as the object whose color is to be changed. For goals like this, two basic interaction patterns can be found in common software.

Choose Tool and Apply. This paradigm requires that the user first decides which tool to use, such as selection or fill color. The tool is then attached to the mouse pointer, often represented by a specific pointer shape. The tool can be applied to the current mouse pointer position with a mouse button click. This pattern is a less obvious variation of a mode switch (Sect. 4.2.4).

Select Objects and Modify. The opposite approach requires that the user first selects the objects that she wants to modify. The objects visually reflect their selection state, for example with a highlighted outline. Then, the selected objects can be modified by direct manipulation or with a click on the appropriate toolbar button.

In Microsoft PowerPoint, both paradigms are mixed and diluted. To add new drawing objects, PowerPoint uses the choose-tool-and-apply pattern with a non-standard behavior: As soon as the new object is created, the chosen tool is immediately released, meaning that multiple inserts require the user to choose the same tool over and over again. On the other hand, for existing objects PowerPoint follows the select-objects-and-modify pattern.

During my field study (Chap. 2) I could observe a meta pattern that was followed by most users to insert multiple similar objects: The first object was inserted as the choose-tool-and-apply pattern suggests. This object then served as a master copy: It was brought into shape with some modifications like color, font and shadow. Finally, copy-paste was used to create multiple copies of the master. This meta pattern fits well to the PowerPoint patterns for new objects, modifying objects and duplicating objects. However, it remains unclear if the PowerPoint UI really fits the users' needs, or if the users chose that meta pattern because it works around the many interactions that would otherwise be required.

In order to facilitate the transition from regular PowerPoint to a new environment, for my prototype I decided to stick with the PowerPoint way of inserting

and modifying elements. However, I expect that a future version of the think-cell software, if based on ongoing user studies, will implement more intuitive and less interaction-intensive interaction patterns for the same tasks.

4.2.6 Feedback to the User

Besides input *from* the user, feedback *to* the user is the other component in human-computer interaction. The challenge of providing good feedback is to give exactly the information required for the user's task, to prioritize that information to prevent information overload on the user's part, and to render the information such that it gets as much of the user's attention as required without unnecessarily distracting the user from her main task.

Although in theory input from the computer to the human could use any of the five senses, the only feedback devices available on a standard personal computer are the screen and the speaker. Yet, for the present application I refrain from using audio as a feedback channel, because my study showed that the target user group works in open-plan offices (Sect. 2.1.2). In this situation, audio feedback from various workplaces that are located next to each other would do more harm than good. In effect, the only feedback channel available for my application is the screen.

The focus of the application is semiautomatic layout based on implicitly specified constraints. Hence, I must provide enough feedback for the user to understand why a certain layout was chosen and which constraints must be added, removed, or changed in order to achieve the desired layout. The visualization of constraints and their impact on the result plays an important role for the learnability, robustness and efficiency of the system: The user must understand the input-result-relation to make appropriate use of the layout tool. Moreover, some kind of preview that explains what would happen if, for instance, the mouse button was clicked in a certain position, together with instant feedback on user actions, helps the user to avoid or at least to notice erroneous input and to learn the optimal interaction patterns.

It might be interesting to incorporate some animation to explain layout changes. In research, animation has been successfully applied in similar settings, where state transitions would otherwise be obscure to the user (e. g., [Gle92]). However, animation requires extra efforts in terms of programming and runtime resources, and it very strongly attracts attention, distracting from anything else the user might be doing. For these reasons, animation must be used with care and I did not use animation at all in the present prototype.

Other uses of feedback are understood in recent office software and should be continued in my application. For example, the 3-dimensional appearance of a button conveys the affordance to push it. The shape changes on mouse-over to confirm its responsiveness and on button-click to acknowledge the recognition of a command. Another example is the mouse pointer shape that reflects system state (an hourglass when the system is temporarily not responsive) and input modes (a cross hair for inserting new objects into a slide). The mouse pointer shape is especially

convenient to encode this kind of information, because this is where the user's focus of vision is most of the time. Therefore, the information comes to the user's attention immediately without distracting from the visual context.

4.3 Specifying the User Interface

Now that I have presented a constraint-based concept for abstract specification of slide layout (Sect. 4.1) and have reviewed the available interaction techniques (Sect. 4.2), in this section I apply the interaction techniques to enable the smart grid concept.

4.3.1 Inserting a New Element

In current PowerPoint, there are two slightly different interactions for inserting new elements. In both cases, the desired element (box, circle, etc.) is selected from the toolbar by clicking the associated button. The button indicates the mode switch: it stays pressed. The mouse cursor changes to a cross hair shape.

In the first case, the user simply clicks at some position within the slide. The selected shape is then inserted at a standard size with its upper left corner positioned at the mouse cursor. In the second case, the user specifies a rectangular area by drag-and-drop. The inserted shape is then stretched to fill the selected area. In both cases, the associated toolbar button leaves its pressed state immediately after the element is inserted and the mouse cursor changes back to the default arrow.

Strangely, more complex objects including charts and tables are immediately inserted with a default size at a default position, when the respective button is clicked. Any desired placement other than the default location requires the user to move and resize the object after insertion. This inconsistent behavior is probably due to implementation details of OLE objects, which should better be hidden from the user in favor of better usability. The Microsoft way to represent OLE objects in the user interface is discussed in section 4.2.4.

My interaction pattern for the insertion of new elements is based on the insertion of native PowerPoint shapes (Fig. 16): The user chooses a smart element from the think-cell toolbar (1) and the mouse cursor turns into a cross hair shape. Again, there are two ways to specify the new element's location: Single-click (2a) and drag-and-drop (2b).

There are ways provided to leave the insertion mode in case that it was entered by mistake. The user may chose another element to insert, either from the think-cell toolbar or from the ordinary PowerPoint toolbar. Alternatively, she can leave the insertion mode altogether by pressing the ESCAPE key. Again, the insertion mode is implicitly left immediately after the new element is inserted. This behavior is questionable but was inherited from PowerPoint for consistency.

Finally, after the insertion the bindings of shapes to gridlines are updated, new gridlines are created where necessary (3) and the solver is triggered to update the over-all layout (4).

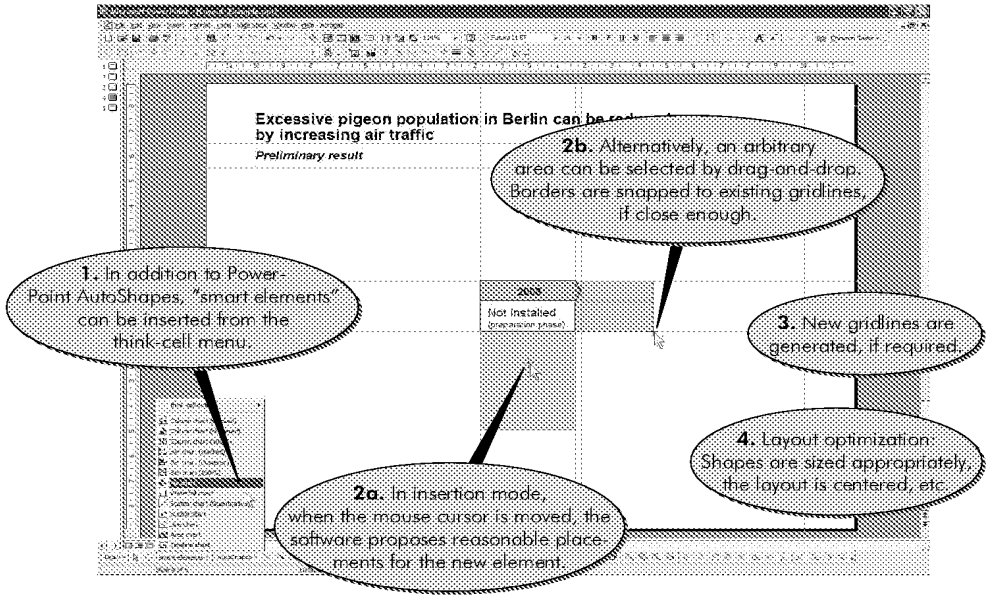


Figure 16: Inserting a new shape into the smart grid

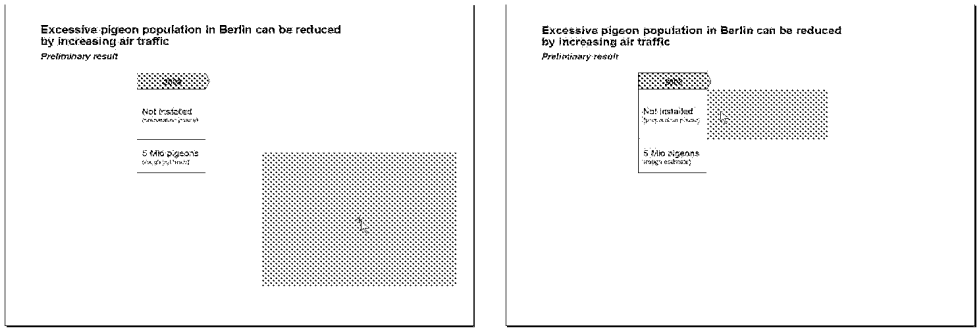
Single-Click Insertion

The single-click insertion is strictly optimized to make the common case fast. The idea is to detect common cases based on the mouse pointer position and existing shapes on the slide. The most typical insertion location that is detected by my software is shown to the user by a transparent rectangle that serves as a placeholder for the new element. If the user is satisfied with the system's proposition, a single click suffices to insert the new element.

This concept is enabled by the smart grid and the solver. In plain PowerPoint, it would be pointless to suggest one pixel position or another. However, with the solver taking care of the details, the space of distinct solutions for placement is drastically reduced: Each edge of the new element's bounding box can either be bound to an existing gridline, or it creates a new gridline between two existing ones. As long as all edges that share the same position are bound to the same gridline and the total order of gridlines is preserved, the absolute placement in terms of pixels does not make a difference. The solver would fix it, anyway.

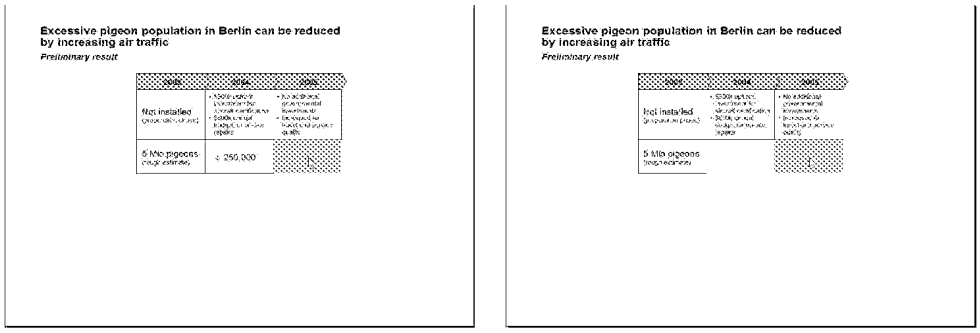
Figures 17 through 20 show the effect of some heuristics that propose common placements. All figures are live screenshots from my prototypical implementation that is presented in chapter 5.

The proposed placement is based on the dimensions of the nearest shapes that are found in the horizontal and vertical projection of the mouse pointer position. When no shapes are found, some arbitrary default size is proposed, centered around the mouse pointer (Fig. 17(a)). On the other hand, if some shape is found and the mouse pointer is sufficiently close to that shape, the new element is placed flush next to it, taking on the same height or width, respectively (Fig. 17(b)). If in both



(a) Suggesting a default placement ... (b) ... or the size of a neighboring shape.

Figure 17: The proposed placement for the new element depends on the shapes that are found in the projection of the mouse pointer position.

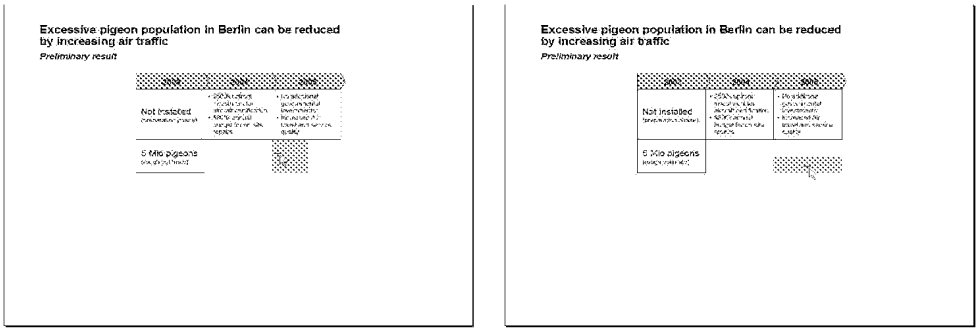


(a) Neighboring shapes suggest their sizes ... (b) ... no matter how close they are.

Figure 18: If there are shapes available in both directions from the mouse pointer, the size of the new shape is entirely determined.

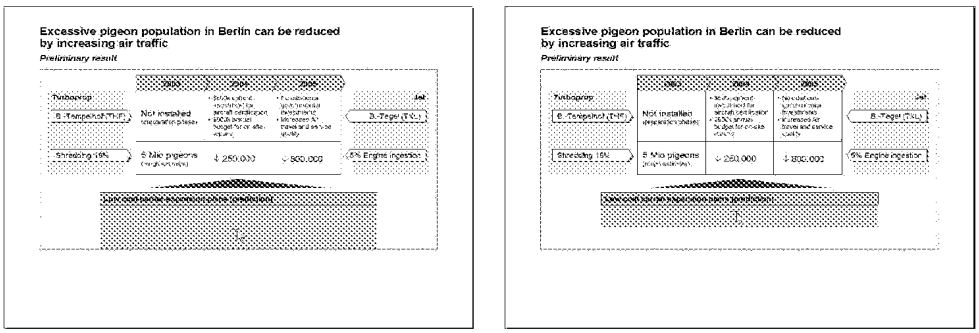
dimensions – horizontally and vertically – a shape is found on the same level as the mouse pointer, the width and height of those shapes are combined to suggest a placement for the new element. This heuristic applies even if those other shapes are found in some distance from the suggested placement (Fig. 18).

As explained above, the only distinct variations of placement in the smart grid are to bind an edge to an existing gridline, or to place it somewhere between two existing gridlines. Both variations are supported by single-click suggestions: When the mouse cursor gets close to some border of the proposed placement, the placement shrinks to take up only half of the suggested place. In this case, three edges of the new placement are still aligned with existing shapes on the slide, while the fourth edge is placed between the gridlines that are imposed by the existing shapes. In any case, the new placement is perfectly aligned without additional efforts from the user's part. (Fig. 19 and 20)



(a) Halving the insertion works for all edges ... (b) ... of the proposed bounding box.

Figure 19: Moving the mouse pointer near the edge of the proposed insertion area of figure 18(b) cuts the area in half.



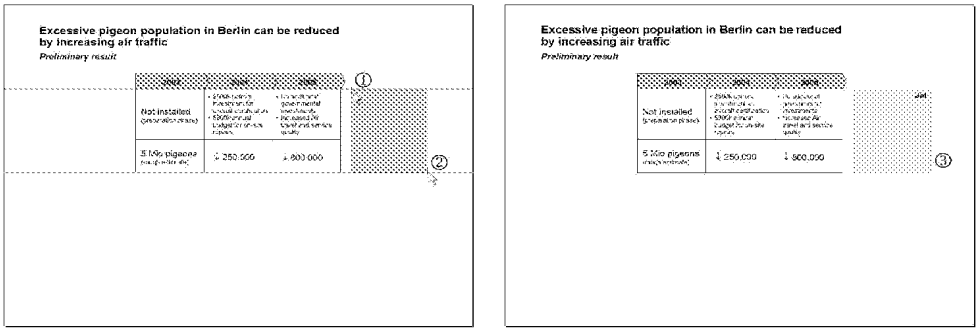
(a) The new element can fill an entire area ... (b) ... or leave a gap to the next shapes.

Figure 20: By simply moving the mouse cursor, many reasonable placements can be realized with a single click.

Drag-and-Drop with Snapping

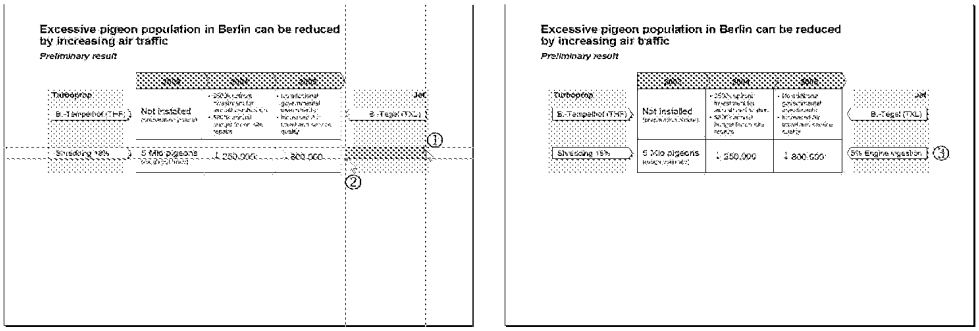
Drag-and-drop requires more user interactions and more attention, therefore taking more time and effort than single-click insertion. However, while with single-click only predefined placements can be chosen, with drag-and-drop the user is free to choose an arbitrary rectangular placement. The drag-and-drop insertion behaves the same like in ordinary PowerPoint, with the only exception that the caret is snapped to existing gridlines. That way, the smart grid provides guidance to support effortless and yet precise alignment with existing shapes. Again, only the placement relative to existing gridlines is relevant; as before, the details of sizes and proportions are up to the solver.

Figures 21 and 22 show, how snapping can be used to align one or more edges of the new element. As before, these figures are live screenshots from my prototype (Chap. 5) with only little touch-up for better understanding. The circled numbers indicate the steps of the interaction: At (1) the left mouse button is pressed. Then, with the button down, the mouse is dragged to position (2), where the button is released. Step (3) shows the result of the interaction, which is meant to be further adjusted by the solver.



(a) User interaction ... (b) ... and the resulting layout

Figure 21: Gridlines highlight to indicate snapping.



(a) User interaction ... (b) ... and the resulting layout

Figure 22: Snapping can happen in both dimensions at the same time and can align the new element with distant shapes.

Resolving Overlaps

Since the solver is meant to interactively rearrange the over-all layout in an attempt to make optimal use of the available slide area, it will be a common case to insert a new element in a place that is already occupied by another shape. Figures 23 through 26 illustrate some of the heuristics I use to resolve overlap that is specified by single-click or drag-and-drop insertion. Three possible intentions must be distinguished.

Replacement. When the new element has exactly the same size and position as some other shape on the slide, that other shape is replaced. Any contained text from the original shape is copied to the new one.[9]

Displacement. When the insertion area precisely overlaps an existing shape in one dimension and only partly overlaps in the other, one edge of that shape is displaced to make room for the new element. While the example shown in figure 23 is a montage, pretending that the solver worked to rearrange

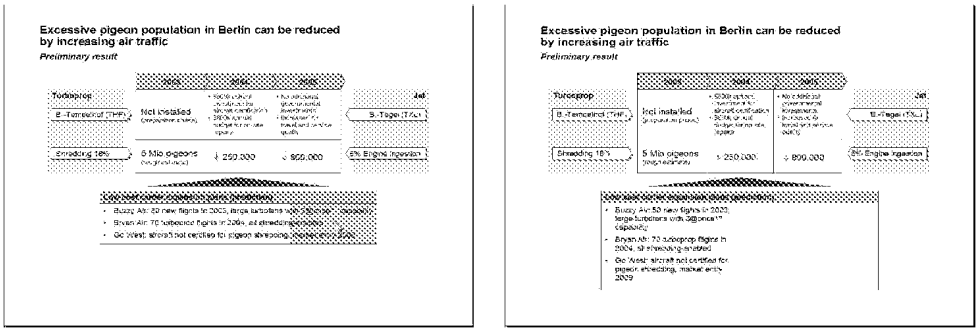
[9] In the current prototype, the placement, replacement and displacement of shapes is already working as specified. The details of handling contained text and other attributes are not yet available, because the think-cell API to PowerPoint still lacks some functionality in this regard.

the layout, figures 24 and 25 are again live screenshots with only minimal touch-up.

Overlap. In any other case, the overlap is taken "as is", leaving all existing shapes in their location and generating new gridlines where required. (Fig. 26)

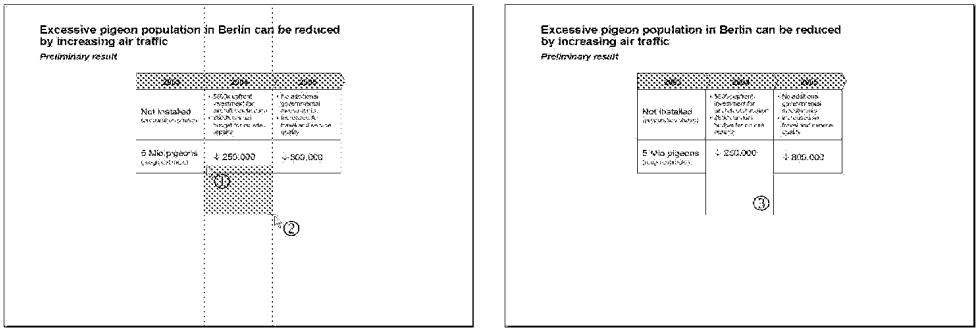
I could use some modifier key to let the user determine whether to choose overlap or displacement. However, this feature is not currently implemented – partly because the usual modifier keys already have some meaning in PowerPoint and partly because it is not yet clear if such explicit user input is required in practice.

Also, in the current implementation feedback is limited to shading the suggested or selected insertion area and highlighting gridlines that snap the caret. In a future version, some kind of advance indication should be provided whether the insertion area is interpreted as overlapping or displacing.



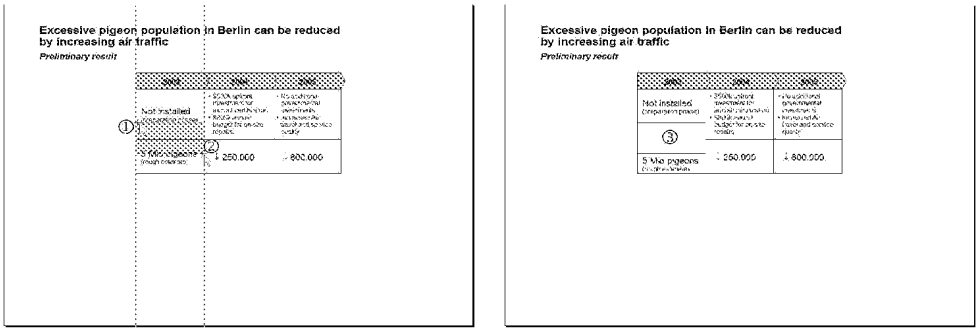
(a) User interaction ... (b) ... and the resulting layout, assuming that the solver has appropriately rearranged the grid.

Figure 23: Any cell of the smart grid can be split with a single click to make room for a new element.



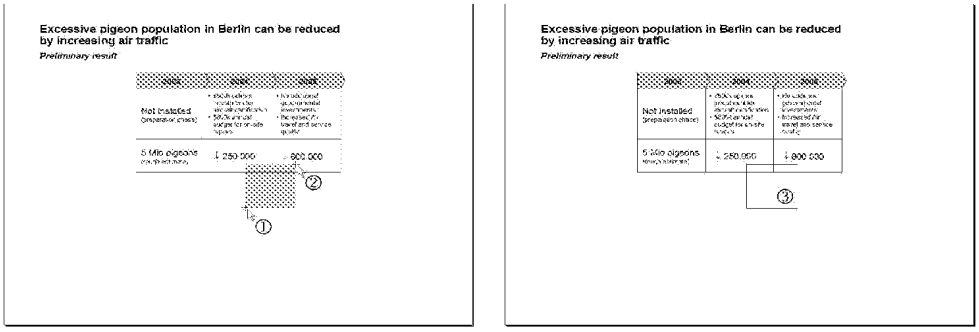
(a) User interaction ... (b) ... and the resulting layout

Figure 24: If the new element precisely overlaps an entire dimension of an existing shape, one edge of that shape is displaced.



(a) User interaction ... (b) ... and the resulting layout

Figure 25: Displacement also works to create a gap between neighboring shapes.



(a) User interaction ... (b) ... and the resulting layout

Figure 26: If the edges of the new element do not coincide with the edges of overlapping shapes, the new element is placed regardless of the overlap.

4.3.2 Inserting Multiple Elements

The insertion of multiple elements at a time is different from the sequential insertion of multiple elements one by one. In the latter case, the placement of an element can be specified as described in the previous section, then the solver updates the layout, then the user determines the next placement, and so forth. In contrast, with paste or move multiple elements must be placed by a single user action, all at a time, and the placement of each element depends not only on the existing layout, but also on the placement of the other shapes that are being pasted.

Just as with single element insertion, the goal is to bind all the inserted shapes to the right gridlines – or generate new gridlines, if required – with as little effort from the user's part as possible.

Copy/Paste and Move

In ordinary PowerPoint, a move operation requires that the desired shapes are selected first. Then, a drag-and-drop operation is performed to move the selected shapes around on the slide. The operation behaves as if the selected shapes were implicitly grouped: Depending on the active settings, the bounding box of the

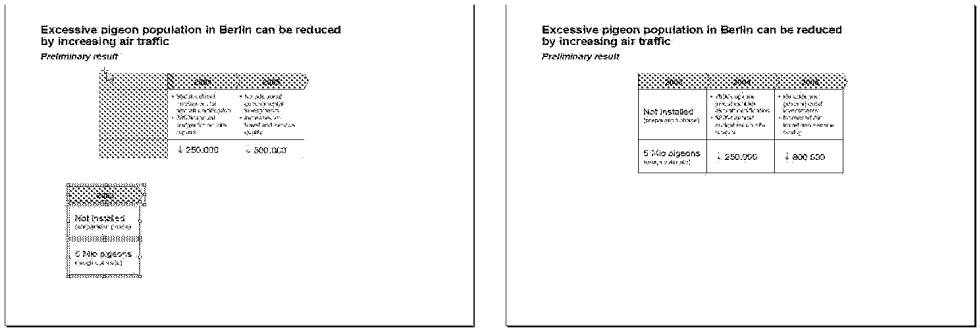
composition is snapped to the regular grid or to other shapes, or is simply moved pixelwise. In any case, all sizes, proportions and relations of the participating shapes remain untouched.

The original PowerPoint paste operation doesn't allow the user to specify the paste location. Objects from the clipboard are inserted at a default location: Shapes that stem from a different slide are inserted at their original location while shapes that originate on the same slide are placed with a little offset to the right and to the bottom.

For insertion into the smart grid I decided that move and paste should appear similar to the user: In both cases, a shadow is attached to the mouse cursor that can be placed with a single click. The shadow indicates where each of the selected elements would go. This approach is much more appropriate for the smart grid than the original PowerPoint usage of move and paste, because it allows the elements to snap to existing gridlines, thus implicitly ensuring alignment with the existing layout.

In my prototype, a dynamic programming algorithm is used to determine the optimal placement of multiple elements. The mouse cursor is assumed to be at the upper left corner of the elements' bounding box. Based on the mouse cursor location, each element is bound to appropriate gridlines, if there are any. To achieve an optimal insertion, elements may be stretched and shrunk. The cost function of the dynamic programming algorithm ensures that gridlines are matched without too much distortion of the elements.

Figure 27 shows an example of a composition of elements being moved (or cut and pasted) together. Thanks to snapping, a somewhat sloppy mouse cursor movement is tolerated. Since the moving elements are of the same types as the shapes next to the intended placement, there is a very strong preference to match those shapes' gridlines. After the insertion, the solver rearranges the smart grid in order to accommodate for the required size of the inserted elements. The screenshots are taken from my prototype, but are manually arranged to simulate the behavior of the solver.



(a) User interaction ...

(b) ... and the resulting layout (after update by solver)

Figure 27: When multiple elements are inserted, a dynamic programming approach ensures optimal fit into an existing layout.

Several variations to the interaction pattern for insertion of multiple elements are conceivable. For example, there is no inherent need to associate the mouse cursor position with the upper left corner of the insertion bounding box. When a cut, copy, or move operation is initiated, a user-specified reference point inside or even outside the bounding box could be saved and used for the insertion placement; Autodesk's AutoCAD [Aut] uses a similar technique for moving and copying objects.

In an attempt to give the user more control over the placement, it should also be possible to specify not only a single mouse cursor position, but again a rectangular screen area. The elements being inserted would then have to arrange within the user specified destination area. Both of these ideas are actually part of my concept but to date could not be realized due to time constraints.

Clone-Insert

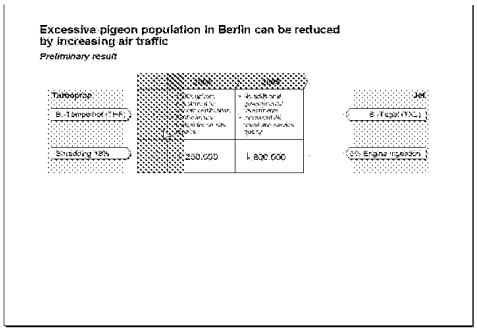
My user study (Chap. 2) revealed typical use cases for the duplication of elements.

- When multiple shapes of the same type and formatting are required in a layout, users typically insert one element and apply all necessary styles and formatting. Then, they duplicate this element to produce the number of elements they need. (see also Sect. 4.2.5)
- Due to the shortcomings of built-in PowerPoint tables, tables are typically built from carefully aligned single boxes. Not only is it very time consuming to create tables like this, but it is especially painful to insert new rows and columns later. (see also Sect. 2.1.5, item D)

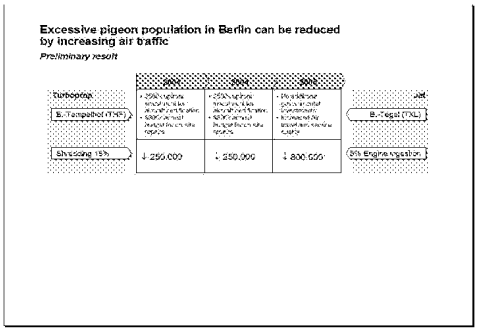
These observations led to a new interaction pattern, which I call clone insert. Because clone insert is very similar to cut, copy, paste operations, I suggest that it is put into the menu next to these operations and its shortcut be CTRL-Y. In its basic application, the clone insert tool copies an element and places the copy aligned and flush next to the original shape, creating a new row or column in the smart grid.

What is more, clone insert not only takes a single element, but automatically determines a group of elements that fill a contiguous range of cells in the smart grid. With a single user action, the entire composition is copied and placed flush and aligned. An example setup is shown in figure 28. While figure 28(a) is a live screenshot with only little touch-up, figure 28(b) is manually arranged, because copying actual shapes is not yet realized in my prototype. However, the determination of joined shapes and the placement of the copy are already working as specified.

Figure 29 shows that in cooperation with the dynamic smart grid, clone insert can do still more: Even in a compact table, entire rows and columns can be inserted. To make room for the new elements, both old and new shapes are temporarily shrunk to fit. Then the solver takes over to rearrange the over-all layout and to provide as much room for each element as required, while keeping all other shapes perfectly aligned.

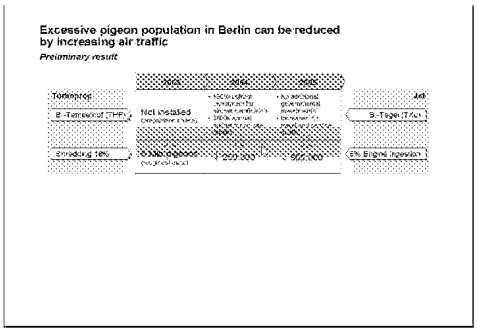


(a) User interaction ...

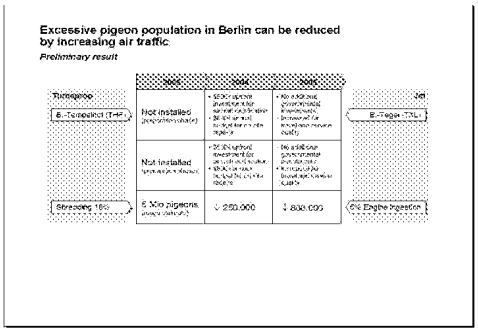


(b) ... and the resulting layout

Figure 28: In clone mode, an entire composition of shapes can be duplicated in a single interaction without the need to explicitly select those shapes.



(a) User interaction ...



(b) ... and the resulting layout (after update by solver)

Figure 29: In combination with the solver, the clone mode is especially convenient to insert entire rows or columns into tables that are built from single shapes.

4.3.3 Selecting Smart Elements

The selection of elements is a part of the user interface that is not primarily targeted at handling the smart grid. However, it is an important part of the *select-and-modify* pattern (cf. Sect. 4.2.5) and therefore we will have a look at some new ideas for selection, which I realized in my software.

First of all, the PowerPoint way to select elements is supported. This includes the exclusive selection of the element that is visible at the mouse cursor position when the left mouse button is clicked without any modifier keys, removing any selections from other elements. When SHIFT is held down for the same action, the selection is toggled for the element at the mouse cursor position, leaving all other selections unaffected. There's a third way to select one or multiple elements in PowerPoint: Starting a drag-and-drop action on a part of the slide where there is no element, creates a *lasso*. The lasso is a rectangular screen area that serves as a bounding box for the elements to select. All elements that are spatially enclosed by the lasso are selected. Also, the SHIFT key can be used to toggle the selection of elements that are captured by the lasso.

Selection Indication

Since smart elements exhibit an enhanced behavior as opposed to native PowerPoint shapes, the user must be provided with feedback to distinguish these two kinds of elements. I chose to show a different style for the outline and the dragging points of selected elements. In addition, a lighter version of the selection indication displays on mouse-over to provide some hint which elements are "smart" and can be selected. Native PowerPoint shapes are not sensitive to mouse-over actions.

While PowerPoint shapes generally have eight dragging points – one in each corner and one in the middle of each side – to modify their size in different dimensions, smart elements offer similar functionality only for the corners. Because of the difference in behavior with regard to the smart grid, I call my dragging points anchorpoints. A detailed description of anchorpoints is given in section 4.3.4 where there are also screenshots that show the appearance of a selected element (Fig. 34).

Outline-based Selection

While a traditional PowerPoint slide is built from basic shapes – rectangles, ovals, arrows, text boxes – and from OLE objects that appear to PowerPoint as simple rectangles, in the smart grid approach the layout exhibits a hierarchical structure. In particular, charts and complex templates are replaced with smart elements. Smart elements are complex compositions of basic shapes. In contrast to traditional PowerPoint templates, smart elements appear to the user as one single PowerPoint object. They configure themselves while interacting with the smart grid, generating and moving their internal shapes as required. In contrast to OLE objects, smart elements are edited "in place" without the need to switch to another application first.[10]

In order to facilitate user interaction with smart elements, I needed to extend the PowerPoint notion of "selection". For purposes like moving, resizing and copying, smart elements can be selected like usual PowerPoint shapes. Any selection of a shape that is part of a smart element, selects the entire smart element itself. On top of that, those shapes of the smart element that offer some configurability present an outline that look different from the PowerPoint selection. These shapes are called *features* of the smart element. The outline of a feature is highlighted in red on mouse-over and turns to blue when it is selected. Just as in PowerPoint, selection is performed with a single click and the SHIFT key can be used to create multiple selections. But in this case, multiple selection is restricted to features of the same type and within the same smart element. In figure 30 you can see a multiple selection of bar chart segments.

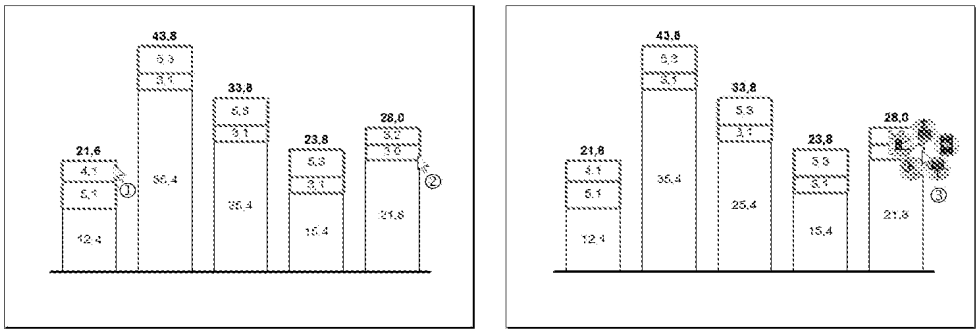
The place to click in order to select a feature is its highlighted outline, not its body as in ordinary PowerPoint. When a mouse click is detected on a feature outline, that click is hidden from PowerPoint to circumvent the traditional selection behavior.

[10] For remarks on mode switch and OLE objects, see section 4.2.4.

Late-breaking result: In recent user studies with the prototype, it turned out that users very much rely on the body of a shape being responsive for selection. Users had great difficulties to understand and use the outline-based selection, and no advantage for the outline-based approach could be observed. Therefore, in upcoming prototypes selection will be implemented to behave exactly like in ordinary PowerPoint.

The Logical Lasso

Multiple selection of smart element features is further supported by a *logical lasso*. Just as PowerPoint's spatial lasso, the logical lasso is used with a drag-and-drop operation. However, in my case logically consecutive features are selected, regardless of their spatial placement. Figure 30 shows live screenshots from my prototype that illustrate the use of the logical lasso: The left mouse button is pushed at (1) and with the mouse button down, the mouse is moved to (2), where it is released. All first and second segments of the bar chart are selected, because the mouse started at the first segment of the first bar and left off at the second segment of the last bar. The context menu for the selection opens on a right mouse button click (3).



(a) The range of selected features is specified by a drag-and-drop operation.

(b) A right mouse button click opens the context menu.

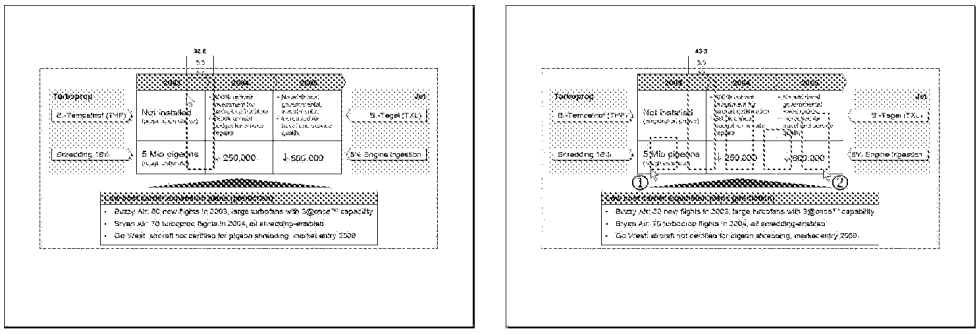
Figure 30: The *logical lasso* allows to select multiple features of the same type within a smart element.

Nearest Neighbor Selection

A common problem of the PowerPoint way to select shapes arises when a shape is hidden behind some other shape. A shape that is not visible on the slide cannot be selected by a single click. Experienced PowerPoint users choose the spatial lasso to catch such a hidden shape.

I propose another solution, which is based on features and their outlines. Based on a strictly 2-dimensional nearest neighbor algorithm, the feature outlines always highlight, even when the associated shape is hidden underneath some other objects. This is illustrated by figure 31: Feature selection and even the logical lasso work the same, regardless of the smart element's visibility. Since the bounding box of a smart

element is also a feature with a responsive outline, my approach can track down smart elements that are hidden in the background of a slide simply by mouse-over.



(a) Outlines of hidden elements highlight on mouse-over.

(b) Selection and multiple selection works regardless of the visibility of the shapes.

Figure 31: A strictly 2-dimensional nearest neighbor approach allows the selection of shapes in the background.

4.3.4 Adjusting the Smart Grid

In sections 4.3.1 and 4.3.2 I explained a range of means that help the user to build a desired layout by implicitly specifying alignment relations when new elements are added. Still, the need will arise to modify the alignment of elements after they are already placed into the smart grid. My concept provides two ways to adjust the smart grid and its contained elements: The user may manually move gridlines and the user may displace the elements' anchorpoints, binding them to new or existing gridlines as desired.

Manually moving gridlines is particularly useful to adjust the global layout of the print space while maintaining alignment relations between elements. For local tweaking – especially in zoom views – it is often appropriate to move anchorpoints directly, breaking up some alignment relationships and creating new ones.

Moving a Gridline. Figure 32 illustrates how gridlines are moved to adjust the over-all slide layout. Each gridline has a handle at each side in the off-slide area of the screen, which displays on mouse-over. Those handles can be dragged to move the respective gridline. The placement of gridline handles avoids confusion with responsive elements on the slide; within the slide area, gridlines can be displayed on request but are not sensitive to mouse interaction. Hence, it is not possible to move gridlines manually in a zoom view, which is also not desirable because moving gridlines has global implications that might not be foreseeable in a zoom view.

The interesting part of moving a gridline is that all edges bound to that gridline are moved accordingly, implicitly resizing their respective shapes. Because minimal sizes of the shapes must be respected, it might not be possible to move a gridline any further at some point. If it turns out to be difficult to understand for the user

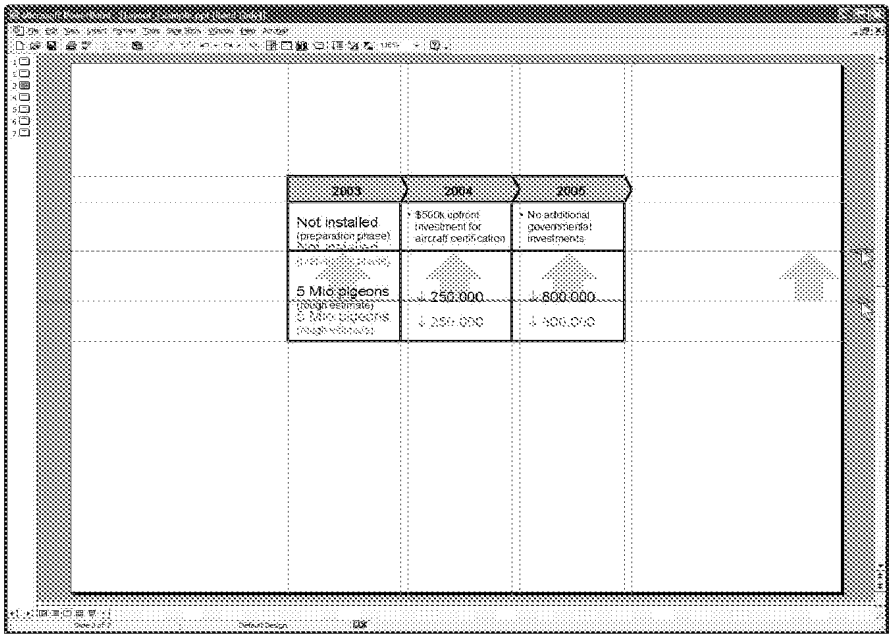


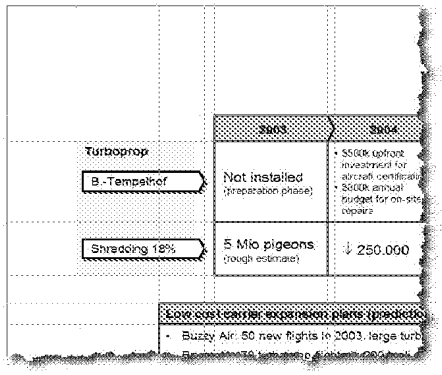
Figure 32: Changing the slide-wide layout while maintaining alignment relations is easily achieved by dragging a gridline

why a gridline gets stuck like that, it might be necessary to add some visual cues for explanation.

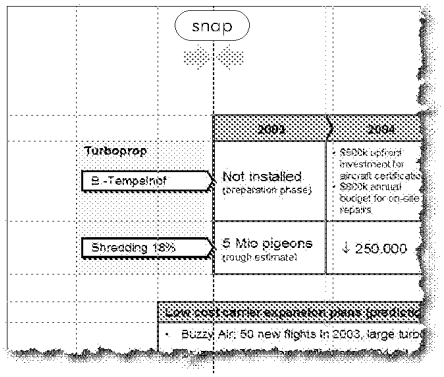
When we have the solver in mind that interactively adapts the entire grid by moving gridlines on its own, there's generally no meaning in moving a single gridline, because the solver would put it back immediately after the user let go. However, there are a few scenarios that require manual movement of gridlines even with the solver in place:

- Gridlines may be *merged* to specify new alignment constraints.
- Gridlines may be *split* to request whitespace between two groups of elements.
- Gridlines could be *pinned* to specify an absolute position constraint (see next paragraph).
- Depending on the implementation of the solver, the ordering of gridlines might or might not specify a constraint for the solver. In case that the solver respects the ordering of gridlines, gridlines could be moved manually to *change the order*.
- As of today, the solver is not yet available. Yet, gridlines are already an effective tool to ease layout with current PowerPoint. Even when the solver is implemented, it might be desirable to have a *fall-back option* in case that automatic layout does not produce the intended results.

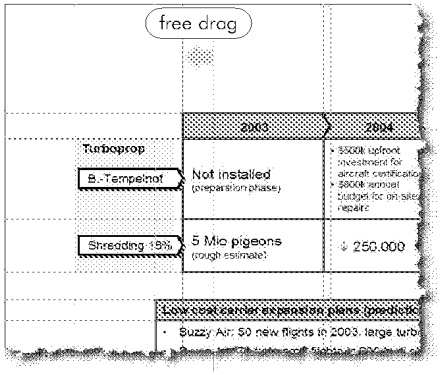
Figure 33 shows a consecutive sequence of gridline actions that illustrate the features that are currently implemented in my prototype. When one gridline is dropped at the exact position of another one, those gridlines are merged, meaning that only one gridline is left which binds all of their edges. Hitting a gridline while moving another gridline is easy, because the moving gridline snaps to other gridlines' positions when it comes close. On the other hand, a gridline may be split up into two gridlines by holding down the CTRL key while dragging it. This operation binds all objects left of the original gridline to the left gridline and binds all objects right of the original gridline to the right gridline, effectively inserting a gap (similarly for horizontal gridlines).



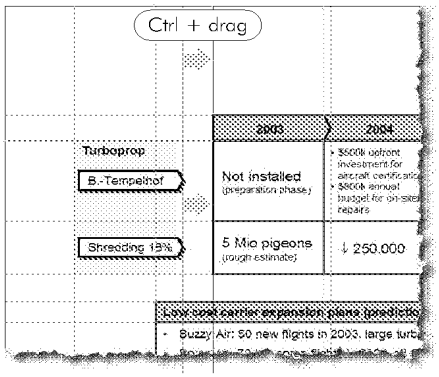
(a) Initial layout



(b) Dragged gridlines snap, merging with the gridline at the drop location.



(c) Further movement of the gridline affects all bound shapes.



(d) Hold down the CTRL key while dragging to split a gridline and insert a gap.

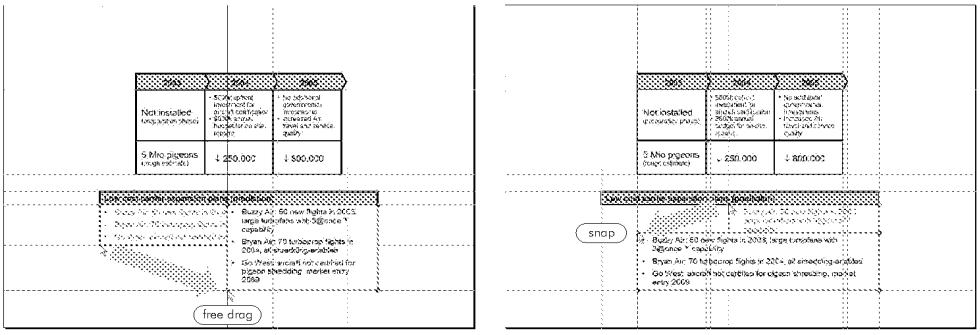
Figure 33: A scenario of gridlines being dragged with different options (live screenshots with annotations)

Pinning a Gridline. The idea of pinning a gridline is to specify an absolute position constraint. For example, the gridlines determining the position of the headline will generally be pinned to ensure that all headlines of a sequence of slides appear at the same position. In addition, the user might wish to pin an arbitrary

gridline if that pushes the solver towards the desired layout. As long as there is no solver moving gridlines around, all gridlines are effectively pinned to the position where the user drags them. Once the solver is in place, any gridline moved by the user that is neither snapped nor split will implicitly be pinned to the position where it is dropped. There will be an option in the contextual menu for a pinned gridline to unpin it again.

Moving an Anchorpoint. As opposed to gridlines, which provide a way for global layout adjustments, anchorpoints are useful to adjust alignment locally (Fig. 34). Anchorpoints are the smart elements' equivalents to the resize handles of native PowerPoint shapes. I use a new name and appearance, because in the context of the smart grid they exhibit an enhanced behavior. (cf. Sect. 4.1.2)

With the solver in mind, moving anchorpoints does not primarily relate to resizing any more, but to specifying alignment relations. Moving an anchorpoint means unbinding it from its gridline and binding it to another one. The user performs the move operation by drag-and-drop. On drop, an anchorpoint always identifies with a crossing of a vertical and a horizontal gridline and snaps to such crossings and to single gridlines while it is dragged. If the anchorpoint is not dropped within the gravity field of a crossing of gridlines, new gridlines are generated as required. After moving an anchorpoint, if a gridline is left with no smart element attached to it, it is removed from the smart grid.



(a) Anchorpoints can move freely ... (b) ... and snap to close-enough gridlines.

Figure 34: Local layout can be adjusted by moving the smart element's anchorpoints. Gridlines are dropped or created as required.

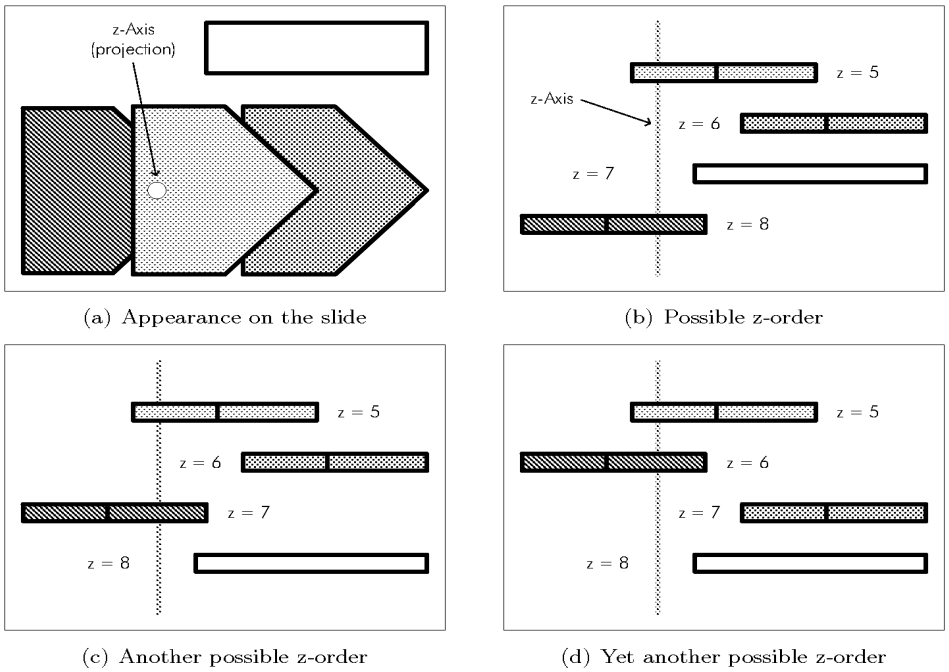
4.3.5 Higher-Order Constraints

Although the grid-based constraints cover a great deal of layout specification, they are not sufficient to specify all inter-shape relations that appear in the context of business slides. Certain constraints simply cannot be expressed by the means of horizontal and vertical gridlines. In this section I will discuss three examples.

Z-Order Relations

By z-order I refer to the third dimension that can often be neglected in slide layout, because slides are always seen "from above" and usually appear just flat. However, when shapes on the slide overlap, it is important to define a z-order relationship. The examples in figure 35 show how z-order relations are implemented in PowerPoint: Each shape on a slide is assigned a unique z-order value, no matter if an overlap occurs or not. As a result, the underlying z-order is unclear from the user's point of view. Any of the z-orders shown in figures 35(b)–35(d) lead to the same appearance, which is displayed in figure 35(a). In order to achieve the correct appearance as seen in figure 36(a), the user has to perform a seemingly arbitrary number of Bring forward operations that depends on the internal z-order representation.

I suggest that it should be possible for the user to select some spot, which is interpreted as the projection of a z-axis. Then with a single interaction, the relative z-order of the shapes that overlap at the selected spot should be rotated (or swapped, in the case of only two shapes). Importantly, any other overlapping must not be affected by this operation. To implement this behavior, a graph-like structure must be maintained in addition to the absolute z-order values.



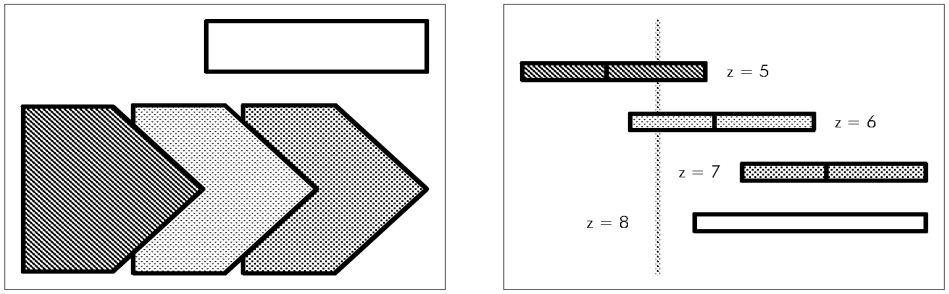
(a) Appearance on the slide

(b) Possible z-order

(c) Another possible z-order

(d) Yet another possible z-order

Figure 35: An example for an arrangement of shapes with wrong z-order



(a) Appearance on the slide (b) "Side view"

Figure 36: The same example as in Fig. 35 with the z-order corrected

Relations between Non-Neighboring Elements.

If two shapes on a slide do not share common gridlines, with the gridline approach alone it is not possible to tell that – for example – both shapes should have the same size. I suggest to implement a number of such constraints, including *same size*, *same formatting* and the like, as drag-and-drop objects. When the user drags a constraint object from one shape and drops it onto another shape, a constraint between those shapes is established in the data model and is visually represented by a colored hairline between the shapes. Badros *et al.* [BNB00] suggest a similar representation for constraints between windows in a WIMP environment.

Element-Specific Constraints

Some more complex smart elements have specific constraints or settings that do not apply to other types of shapes. In the current prototype, each smart element has a contextual menu associated with it (Figs. 15 and 30(b)). A right mouse button click opens the contextual menu of the selected smart element or elements. Depending on the type of the smart element, the contextual menu offers buttons for color, font size and the like. In the case of charts, for example, there are also options to turn labels and other specific features on or off.

5 Implementation

The prototype that was developed in the course of this work is based on a product by think-cell Software. The think-cell product is an add-in to Microsoft PowerPoint that uses a variety of techniques to interact with PowerPoint (Fig. 37). The *Microsoft Office Object Model* and the *Active Document Interface*, which are designed for add-ins like the think-cell product, allow only limited access to the PowerPoint engine. Therefore, in the think-cell implementation, these techniques are supplemented by *Window Subclassing* for message interception. Moreover, think-cell uses *DirectX* to create custom on-screen UI widgets.

My implementation does not make immediate use of any of the techniques mentioned. The classes that I implemented merely interact with the high-level C++ programming interface of the think-cell product and are compiled as part of that product.

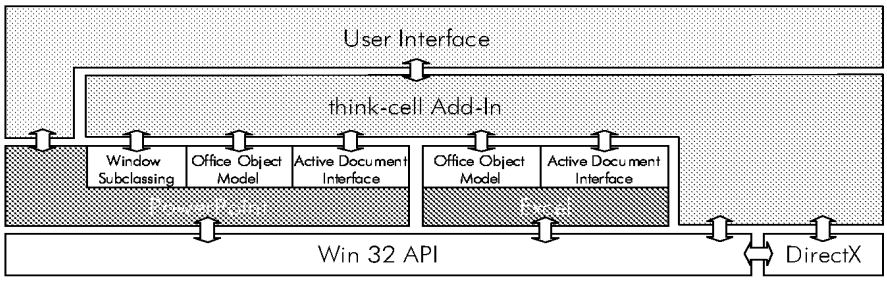


Figure 37: The think-cell architecture uses a variety of techniques to interact with PowerPoint.

5.1 Program Architecture

To get an understanding of the program architecture in general and the user interface code in particular, in this section a number of structural diagrams are discussed in turn. I begin with an overview of the major entities and relations, followed by a close-up look at a hierarchy of UI-related subclasses and finally I examine some interaction scenarios involving a couple of classes.

5.1.1 Overview: Entity-Relation

Figure 38 provides an overview of the most important inter-class relationships of the C++ code. The **CPPTFrame** class relates to a runtime instance of PowerPoint. It contains presentations and document windows. PowerPoint is an MDI[11] application and each instance of **CPPTDocWnd** relates to a runtime document window. Both classes intercept *Windows* messages, most prominently repaint requests and user input like mouse movements and key presses. They process and forward those messages at application or window level, respectively.

[11] Multiple Document Interface (MDI): A single running application instance can open and edit multiple independent documents at a time.

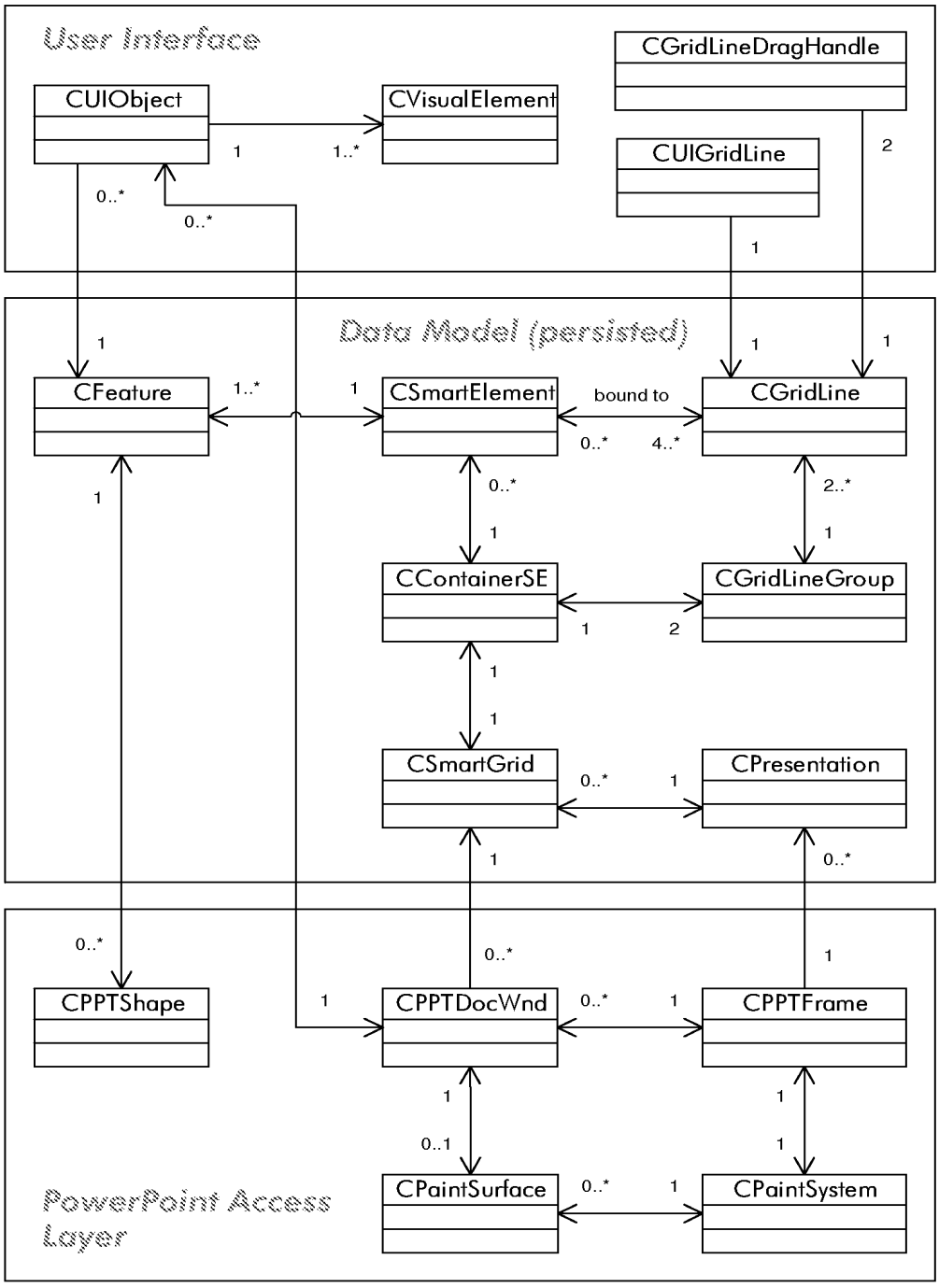


Figure 38: Program architecture – relations between selected classes

The interface to PowerPoint is complemented by the **CPPTShape** class. There is an instance of this class for each shape – rectangle, arrow, text box and the like – in a PowerPoint document. The methods of this class provide ways to add, modify and displace PowerPoint shapes.

While each document window contains precisely one presentation, the same presentation may be opened in multiple document windows simultaneously. Hence, presentations are represented independently from document windows by the class **CPresentation**. Presentation objects contain static or user specified information about a PowerPoint document. Each presentation consists of a number of slides, which in our data model are represented by instances of **CSmartGrid**. The smart grid is the link between a presentation and its smart elements. Because a document window can edit one slide at a time, each instance of **CPPTDocWnd** needs only a single reference to the smart grid that is currently visible.

A priori, each slide has a smart grid and a container, which is an instance of **CContainerSE**. The container spans the entire slide and holds gridlines to represent the positions and spatial relations of its contained smart elements. The container itself is invisible to the user and indifferent against user action.

Gridlines, being represented by **CGridLine** objects, are partitioned by their orientation – vertical and horizontal – and held by one of two **CGridLineGroup** objects per container. Each group of gridlines always contains at least two gridlines, which indicate the lower and upper bounds of their container in the respective dimension.

Smart elements – i. e., instances of **CSmartElement** – are placed relative to a slide's container, their location being determined by the gridlines they are bound to. Each smart element represents a self-contained composition of arbitrary complexity that is placed and sized in its entirety. An example for a complex smart element is a chart as exemplified in figure 30. Each smart element is made up of one or more features, which in the case of a chart correspond to columns and labels and are represented by the class **CFeature**. Features usually have user specified options, but cannot be moved or resized independently from their smart element. A feature, in turn, usually consists of one or more PowerPoint shapes. However, certain features serve only mediating purposes in the user interface and do not have a corresponding PowerPoint shape.

In fact, the objects which I call features are essential for the user interface. On request, a feature generates a number of instances of **CUIObject** – handles or menu items, for example – that provide means for the user to modify the feature (cf. Sect. 5.1.3). All UI objects of all currently visible features are registered with the document window and receive messages therefrom. The visual representation of a UI object involves mapping texture coordinates to screen coordinates and is implemented in the class **CVisualElement**.

This same pattern applies to the user interface representation for gridlines. To be visible and user-modifiable, **CGridLine** objects generate instances of **CUIGridLine** and **CGridLineHotDragHandle**. Indeed, these latter classes are subclasses of

CUIObject (cf. Fig. 39(a)) and as such also own visual elements and are message-driven by the document window.

5.1.2 Hierarchy and Interfaces of UI Classes

Some selected class diagrams are depicted in figure 39. The root class **CUIObject** holds a reference to its document window and offers an interface to process all kinds of low level user interface messages. The subclasses realize different UI widgets by implementing the processing of those messages differently. In particular, the subclasses of **CHotZone** offer specific functions that provide a high-level interpretation of the low-level UI input, based on notions like hit test, mouse capture and the like. For example, in the case of the **CHotButton** class, a sequence of the *OnLButtonDown()*/*OnLButtonUp()* mouse messages triggers a call to the *OnClick()* method, if at the times of both events the mouse pointer is found within the UI object's responsive area.

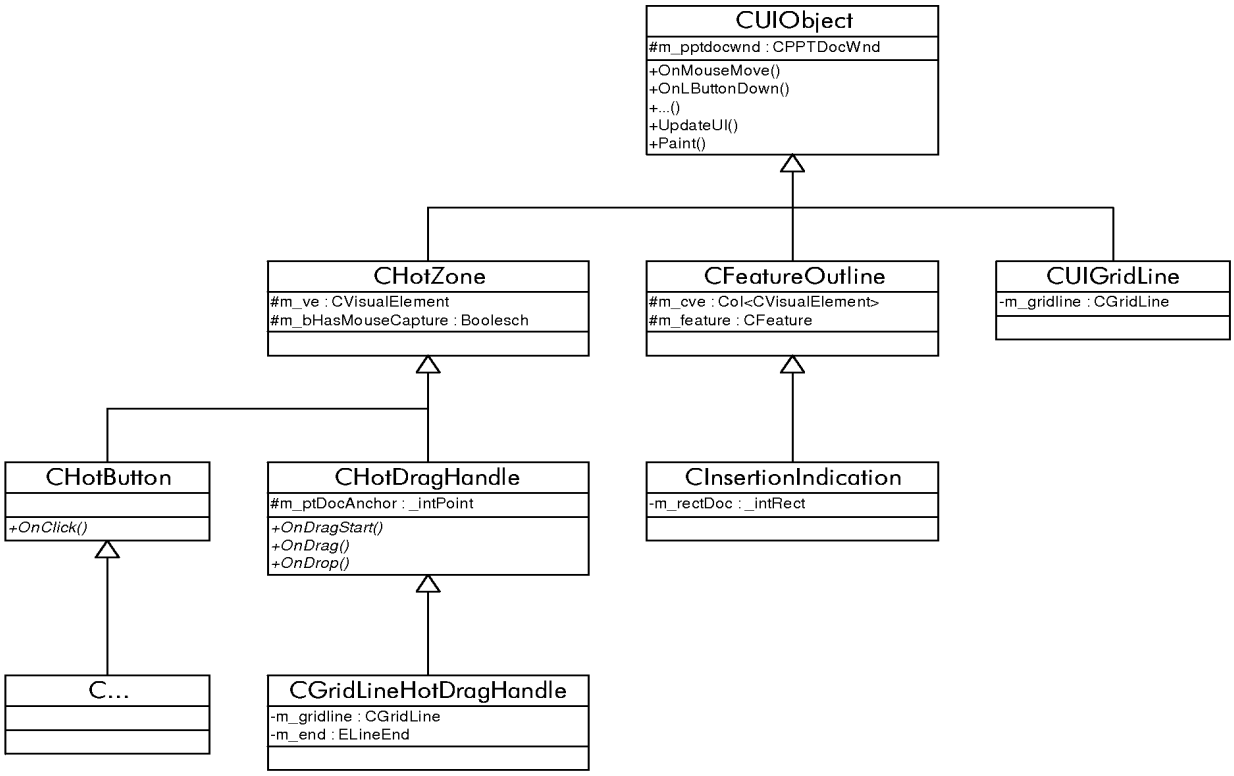
The application specific methods like *OnClick()* are abstract, meaning that the respective class itself cannot be instantiated. To use the functionality provided by the UI classes, each UI widget is programmed as its own subclass, implementing its specific behavior in those abstract methods. Other UI components are too specific to be derived from **CHotZone** or its subclasses. Examples are **CFeatureOutline** and **CUIGridLine**. These classes implement the interpretation of low-level messages themselves, deriving directly from class **CUIObject** (Fig. 39(a)).

The inheritance graph of **CVisualElement** (Fig. 39(b)) partly parallels the hierarchy of subclasses of **CUIObject**. Most UI objects have some visual representation that is made up of one or more visual elements. The methods and attributes of visual elements serve to create an appropriate texture mapping, transforming coordinates from a precompiled texture bitmap to document-oriented coordinates and finally to on-screen coordinates, taking zooming and scrolling into account. The subclasses of **CVisualElement** are tailored to their specific use with the objective to minimize the interface to their respective subclass of **CUIObject**.

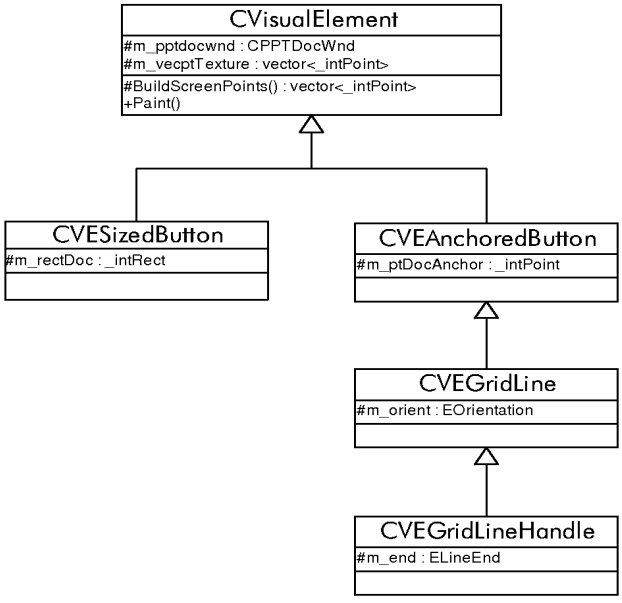
As can be seen in figure 38, features are contained by a smart element and smart elements are contained by a container. Now, figure 39(c)) shows that smart elements also *are* features and that containers also *are* smart elements. The class **CSmartElement** derives from **CFeature**, because the latter provides functionality to show user interface components, and smart elements need to have a user interface, too. On the other hand, the relationship between **CSmartElement** and **CContainerSE** enables hierarchical layouts with gridlines at different levels.

5.1.3 Windows Message Processing

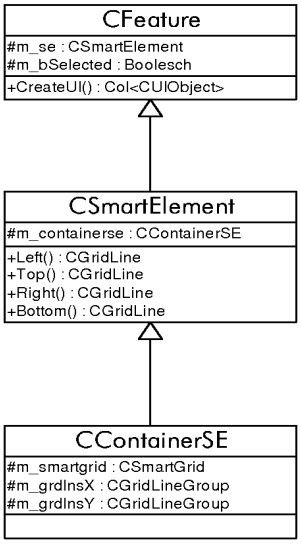
User input processing in my prototype is based on Microsoft Windows messages. In Windows, messages are responsible for inter-process communication. Messages that originate in the operating system are queued in thread-specific message queues and are processed whenever a thread makes a system call to *PeekMessage()* or



(a) UI objects receive and process user input messages from the document window.



(b) Visual elements provide visual feedback to explain the state and effect of UI objects.



(c) Features represent user-modifiable entities.

Figure 39: Selected class diagrams

GetMessage(). The user thread then typically makes a call to *DispatchMessage()*, which is another system routine that effectively calls the *DefWindowProc()* procedure of the concerned window. The *DefWindowProc()* may be overloaded with user code. While a message is being processed, often more messages are generated and sent to other windows or threads. The respective system call is *SendMessage()*, which is implemented synchronously: When the message is sent to the same thread, *SendMessage()* effectively makes a call to the respective subroutine. Otherwise, the message is queued in the message queue of the destination thread and the sending thread is put on hold until the message has been processed.

The prototype uses a number of so-called hooks[12] to get notified when certain messages are retrieved by PowerPoint via *PeekMessage()* or *GetMessage()*. Moreover, using window subclassing, the *DefWindowProc()* of the PowerPoint document windows is overloaded. That way, custom code gets executed whenever a message of interest comes by, which gives the opportunity to realize an interactive add-in to PowerPoint.

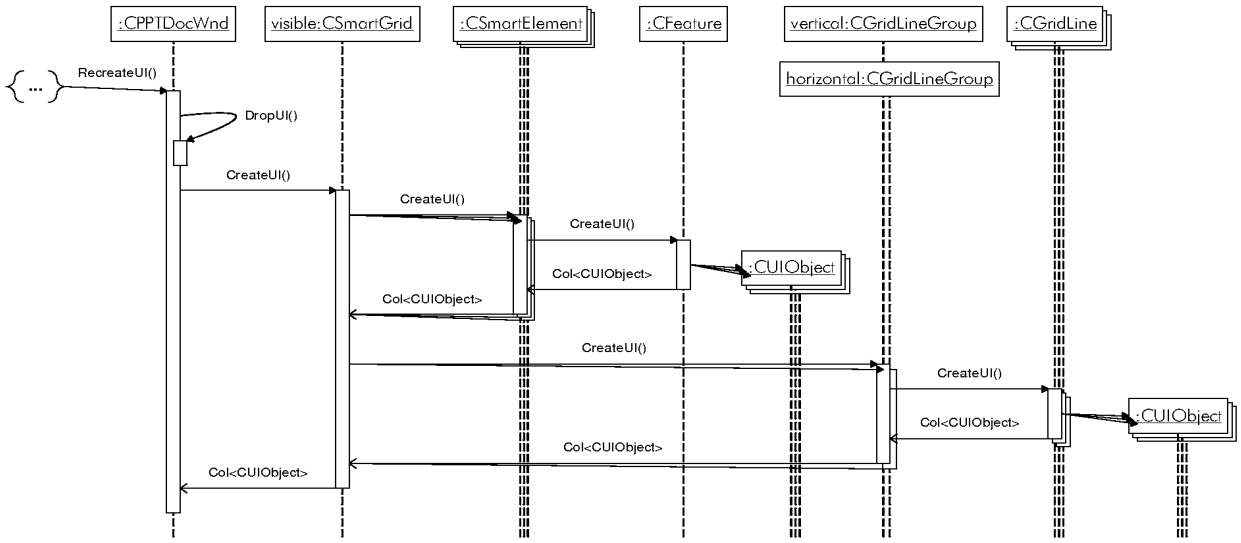


Figure 40: Instantiation of appropriate UI objects on request

For example, in response to some external event, the framework decides to recreate the user interface. Figure 40 exemplifies what then happens: The framework generates a message *RecreateUI()* that is sent to a document window.[13] The document window first drops any existing UI objects and then fetches a collection[14] of newly created UI objects from its visible smart grid. The call is recursively passed down to all smart elements, features and gridlines, that are placed in that smart

[12] In the Microsoft Windows API, a *hook* is a call-back technique that registers user code to be executed at certain system events.

[13] I use the Microsoft terminology here. Read: The *RecreateUI()* method of an instance of class **CPPTDocWnd** is invoked.

[14] **Col<*type*>** is a custom implementation of a collection, which represents a list of objects of the same type that can be iterated.

grid. At each stage, UI objects are generated and are collected from any subordinate levels. The union set of all UI objects is then returned.

Figure 41 gives an idea of how user interaction propagates through the system and finally generates appropriate on-screen feedback. When the framework intercepts a mouse event message, such as *ButtonDown*, the message gets forwarded to the responsible document window and its UI objects. Each UI object has to make a decision if it is concerned by that mouse action, depending on the current mouse pointer location and possibly other UI objects concurrently capturing the mouse. If appropriate, the UI object puts its respective visual elements into another state – say, *highlight* – to reflect its responsiveness. Moreover, the UI object that was hit calls its own *OnClick()* method that finally applies the intended changes to the data model. Meanwhile, the visual element has notified the framework of the invalidation of the screen region where a change in the UI occurred.

Depending on the effect that the mouse interaction had on the underlying data model, the smart grid, its shapes and its UI objects may explicitly be called to update themselves. The propagation of *UpdateShapes()* and *UpdateUI()* messages are not reflected in the sequence diagram; they take similar paths as the *CreateUI()* message in figure 40. At some point in time, there will be a *Paint()* message generated from the framework, which causes the visual elements to update their texture mapping with regard to current document and screen coordinates, and finally reflects the internal state on the screen.

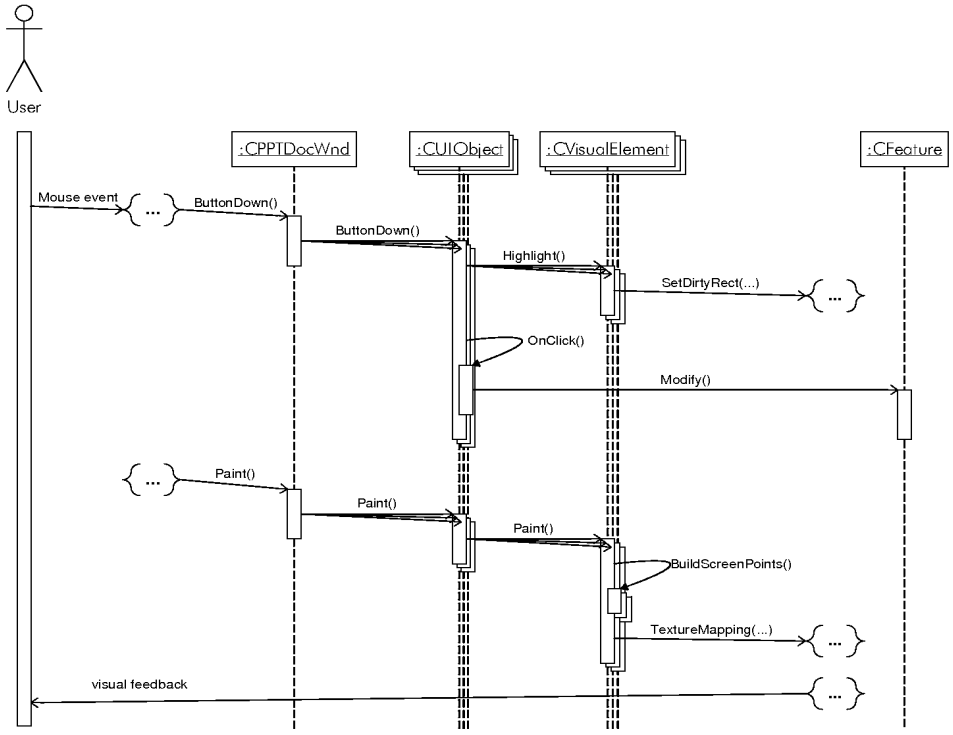


Figure 41: Processing of user input and generation of visual feedback

5.2 An Application of Dynamic Programming

Large parts of my implementation are relatively straight-forward compositions of function calls, such as the control flow that generates a union set of all currently required UI objects (Fig. 40). In some cases, however, there was opportunity to apply results from theoretical computer science. In this section, an implementation of the dynamic programming approach is presented, that serves for optimal simultaneous placement of multiple shapes.

5.2.1 The Gridline Matching Problem

When a slide is composed using Microsoft PowerPoint, common user actions like copy/paste or move can be applied not only to single objects, but also to compositions of multiple objects. In the traditional PowerPoint interface, such compositions show static relationships and are handled just the same way as a single shape. This is fine as long as the placement is purely pixel-oriented.

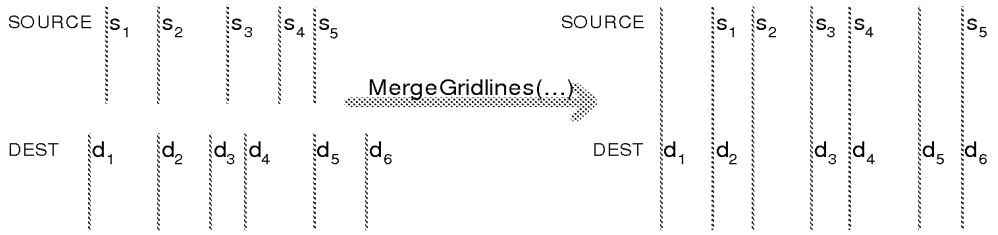


Figure 42: Matching two lists of gridlines

The insertion of multiple elements into the smart grid was discussed in section 4.3.2. The question arises, to which gridlines pasted shapes should be bound. Trivially, the pasted shape or composition of shapes would preserve its internal structure and proportions, and new gridlines would be generated wherever the shapes' edges fell. Since the slide layout is instantly rearranged by the solver, the placement and the proportions of inserted shapes would be adapted, anyway. Therefore, in the context of the smart grid, the precise pixel-based placement of shapes is almost irrelevant. What counts is the order of the gridlines to which shapes are bound. The multiple-placement problem in the smart grid boils down to matching two sets of gridlines, which is illustrated in figure 42: The gridlines from the source location of the pasted shapes may be identified with gridlines at the destination, or may be placed between two consecutive destination gridlines.

This decision might not only affect a single shape, but would have an effect of the overall composition of shapes that are inserted concurrently. For example, shapes that are placed flush next to each other in the original location should probably not be torn apart when they are inserted into another smart grid. On the other hand, some stretching and bending might be necessary to meet the user's intention; imagine a table row whose contents are meant to fit the layout of another table.

I suggest that to a large degree it is possible to foresee the user's intention and place the pasted shapes where they are meant to be. Some key indicators that can help to understand the user's intention are:

- Spatial proximity of edges
- Similarity of shapes in terms of type and formatting
- Patterns that indicate a common case, such as a table row being inserted

These hints can be encoded in a cost function that – given the environment of the source shapes and that of the destination slide – calculates the costs of some source gridline being placed before, on, or after some destination gridline. When every aspect perfectly fits, the costs will be zero; the more counter indication is detected for the placement in question, the bigger the costs will be. Given such a local cost function, the gridline matching problem can be seen as an optimization problem. The optimal solution is a list of SOURCE, DEST and IDENTIFY instructions that describes the order in which gridlines must be taken to create a matching with minimal cumulated costs.

I do not present a detailed cost function here. It is plausible that a polynomial-time cost function is feasible; however, a lot of testing and fine-tuning will be required to achieve reasonable results. For the current implementation I chose a trivial cost function that takes the relative distance between a source gridline and a destination gridline as a measure for the cost, and defines the cost for "very close" to be zero. This algorithm implements a simple snapping behavior, but a much more sophisticated matching is desirable for a future version of the software.

5.2.2 A Recursive Algorithm with Exponential Running Time

The graph in figure 43 illustrates the interpretation of the gridline matching problem as a shortest path problem. A straight forward algorithm to find an optimal solution to this problem is presented in figure 44. The recursive *MergeGridlines()* function takes as input two lists of gridlines and returns the minimal costs along with the optimal path. The algorithm effectively does a complete enumeration of all possible paths from source `s` to target `t`. Assuming that the number of source gridlines n and the number of destination gridlines m are of the same order, this leads to an exponential asymptotic running time of $\Theta(3^n)$: The algorithm does not have any loops and therefore only the recursive function calls contribute to the asymptotic running time. There are three recursive calls altogether, and for about half of all nodes in the shortest-path-graph, all three calls are carried out. I further assume that the cost function has some polynomial running time characteristics.

The cost function in the recursive algorithm is called *CostNext()*. This is, because given a certain state of the calculation, this function calculates the costs for making a SOURCE, DEST or IDENTIFY decision for the next gridlines (i. e., the heads of the *listSource* and *listDest* parameters).

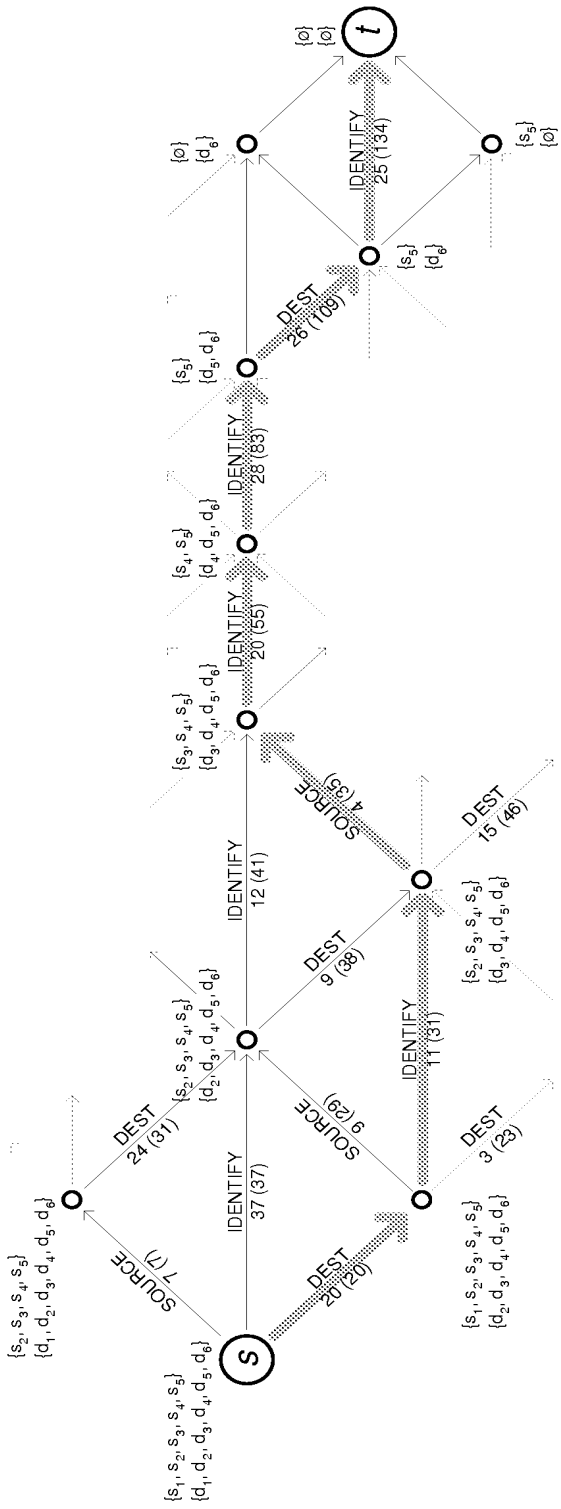


Figure 43: Gridline matching as a shortest path problem (cumulated costs in parentheses), comp. Fig. 42

Algorithm: MergeGridlines (recursive shortest path)

Input: *listSource*, *listDest* : **List of** Gridline

Output: *minimalCost* : Accumulated cost for the optimal path
optimalPath : A list of SOURCE, DEST, IDENTIFY that describes the optimal path

Variables: *costSource*, *costDest*, *costIdentify*, *minimalCost* : Cost
pathSource, *pathDest*, *pathIdentify*, *optimalPath* : **List of** Choice

```
Begin
  costSource ← 0, costDest ← 0, costIdentify ← 0

  If Not IsEmpty(listSource) Then
    (costSource, pathSource) ← MergeGridlines(Tail(listSource), listDest)
                                                  // Tail((a, b, c)) ↦ (b, c)
    costSource ← costSource + CostNext(SOURCE, ...)
  End If
  If Not IsEmpty(listDest) Then
    (costDest, pathDest) ← MergeGridlines(listSource, Tail(listDest))
    costDest ← costDest + CostNext(DEST, ...)
  End If
  If Not (IsEmpty(listSource) Or IsEmpty(listDest)) Then
    (costIdentify, pathIdentify) ← MergeGridlines(Tail(listSource), Tail(listDest))
    costIdentify ← costIdentify + CostNext(IDENTIFY, ...)
  End If

  If costDest < costSource Then
    If costDest < costIdentify Then
      minimalCost ← costDest
      optimalPath ← DEST.pathDest                          // a.(b, c) ↦ (a, b, c)
    Else
      minimalCost ← costIdentify
      optimalPath ← IDENTIFY.pathIdentify
    End If
  Else
    If costSource < costIdentify Then
      minimalCost ← costSource
      optimalPath ← SOURCE.pathSource
    Else
      minimalCost ← costIdentify
      optimalPath ← IDENTIFY.pathIdentify
    End If
  End If
  Return (minimalCost, optimalPath)
End
```

Figure 44: An exponential-time recursive algorithm for gridline matching

5.2.3 Finding the Optimal Solution in Polynomial Time

The analysis of the recursive algorithm reveals that the local costs for each node are calculated up to three times. Also, the graph in figure 43 has a rectangular structure, suggesting that the results of the cost calculations could be cached in a regular matrix. This is precisely what the *dynamic programming* technique describes (see Fig. 45): A matrix is filled with cumulated minimal costs and then the optimal solution is reconstructed from the matrix. To make the reconstruction possible, not only costs must be stored in the matrix, but also the locally optimal decisions.

The respective algorithm is presented in figure 46. The iterative implementation of the *MergeGridlines()* function has the same interface (input and output) as the recursive variant. However, it runs in polynomial time: The loop body is executed $m + n(m + 1)$ times, where m and n are the numbers of destination and source gridlines in the input. Again, I assume the cost function to be of polynomial time complexity. Thus, the asymptotic running time for the dynamic programming algorithm is $\Theta(nm)$ or $\Theta(n^2)$, if we assume n and m to be of the same order.

In this case, the cost function is called *CostPrev()*, because it calculates the costs for a decision that has already been made to reach the current state. The algorithm then stores only the decision that has minimal cumulated cost and disregards the other ones.

Once the matrix is filled, the bottom right corner reflects the total costs of the optimal path, which can now be reconstructed by following the matching decision annotations. The reconstruction of the optimal path is trivial and takes linear time[15], thus not contributing to the over-all asymptotic running time of the algorithm.

Source \ Dest.	1	2	3	4	5	6	Solution
1	**0**	DEST **20**	DEST 23	DEST 77	DEST 165	DEST 317	DEST 428
2	SOURCE 7	SOURCE 29	IDENTIFY **31**	DEST 46	DEST 59	IDENTIFY 178	DEST 210
3	SOURCE 11	DEST 18	SOURCE **35**	DEST 41	SOURCE 61	IDENTIFY 74	IDENTIFY 199
4	SOURCE 26	SOURCE 24	DEST 29	IDENTIFY **55**	SOURCE 79	SOURCE 115	DEST 171
5	SOURCE 58	SOURCE 70	SOURCE 89	DEST 133	IDENTIFY **83**	DEST **109**	DEST 147
Solution	SOURCE 96	DEST 131	IDENTIFY 81	IDENTIFY 97	DEST 105	DEST 139	IDENTIFY **134**

Figure 45: Dynamic programming cost matrix with optimal path reconstruction, comp. Figs. 42 and 43

[15]The asymptotic running time for optimal path reconstruction is $O(n + m)$, because in each run of the *while* loop either i or j or both are decreased.

Algorithm: MergeGridlines (shortest path by dynamic programming)

Input: *listSource* : **Array** [1..n] **of** Gridline
listDest : **Array** [1..m] **of** Gridline

Output: *minimalCost* : Accumulated cost for the optimal path
optimalPath : A list of SOURCE, DEST, IDENTIFY

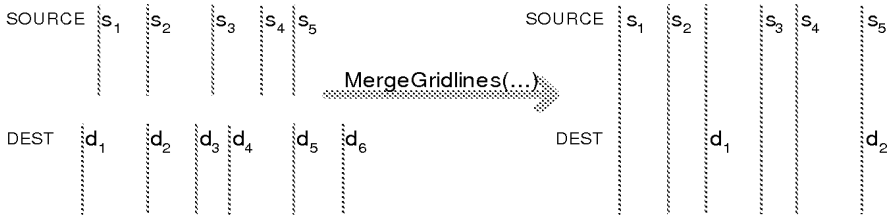
Variables: *costSource*, *costDest*, *costIdentify*, *minimalCost* : Cost
optimalPath : **List of** Choice
matrixCost : **Array** [0..n+1][0..m+1] **of** Cost
matrixChoice : **Array** [1..n+1][1..m+1] **of** Choice
iSrc, *iDst*, *iDstStart* : Index

```
Begin
  Initialize(matrixCost, ∞)
  matrixCost[1][1] ← 0
  For iSrc ← 1 To n + 1 Do
    If iSrc = 1 Then iDstStart ← 2 Else iDstStart ← 1    // Skip upper left corner
    For iDst ← iDstStart To m + 1 Do
      costSource ← CostPrev(SOURCE, ...) + matrixCost[iSrc − 1][iDst]
      costDest ← CostPrev(DEST, ...) + matrixCost[iSrc][iDst − 1]
      costIdentify ← CostPrev(IDENTIFY, ...) + matrixCost[iSrc − 1][iDst − 1]
      If costDest < costSource Then
        If costDest < costIdentify Then
          matrixCost[iSrc][iDst] ← costDest
          matrixChoice[iSrc][iDst] ← DEST
        Else
          matrixCost[iSrc][iDst] ← costIdentify
          matrixChoice[iSrc][iDst] ← IDENTIFY
        End If
      Else
        If costSource < costIdentify Then
          matrixCost[iSrc][iDst] ← costSource
          matrixChoice[iSrc][iDst] ← SOURCE
        Else
          matrixCost[iSrc][iDst] ← costIdentify
          matrixChoice[iSrc][iDst] ← IDENTIFY
        End If
      End If
    End For
  End For
                                                  // Reconstruct optimal solution
  i ← n + 1, j ← m + 1
  minimalCost ← matrixCost[i][j]
  optimalPath ← ()
  While i > 1 Or j > 1 Do
    optimalPath ← matrixChoice[i][j].optimalPath          // a.(b, c) ↦ (a, b, c)
    If matrixChoice[i][j] = SOURCE Or matrixChoice[i][j] = IDENTIFY Then
      i ← i − 1
    End If
    If matrixChoice[i][j] = DEST Or matrixChoice[i][j] = IDENTIFY Then
      j ← j − 1
    End If
  End While
  Return (minimalCost, optimalPath)
End
```

Figure 46: A dynamic programming algorithm for gridline matching, running in polynomial-time

5.2.4 Advanced Gridline Matching Considerations

In practice, there are a lot more issues involved with gridline matching than the basic matching discussed in the previous sections. For example, which exactly is the list of destination gridlines? Depending on the user input, some subset of the available gridlines from the destination slide must be chosen. The precise semantics of mouse input – typically a point or a rectangle – must be defined. In case of a single point, it is unclear where the input of destination gridlines stops. One possible implementation would take all available destination gridlines and treat the entire last row of the cost matrix as valid solutions. Figure 47 exemplifies the optimal solution in this case, where the minimum cost value from the last row of the matrix is taken as the starting point for the reconstruction.



(a) Gridlines being matched

Source \ Dest.	1	2	3	4	5	6	Solution
1	**0**	DEST 20	DEST 23	DEST 77	DEST 165	DEST 317	DEST 428
2	**SOURCE 7**	SOURCE 29	IDENTIFY 31	DEST 46	DEST 59	IDENTIFY 178	DEST 210
3	**SOURCE 11**	**DEST 18**	SOURCE 35	DEST 41	SOURCE 61	IDENTIFY 74	IDENTIFY 199
4	SOURCE 26	**SOURCE 24**	DEST 29	IDENTIFY 55	SOURCE 79	SOURCE 115	DEST 171
5	SOURCE 58	**SOURCE 70**	SOURCE 89	DEST 133	IDENTIFY 83	DEST 109	DEST 147
Solution	SOURCE 96	DEST 131	**IDENTIFY 81**	IDENTIFY 97	DEST 105	DEST 139	IDENTIFY 134

(b) Optimal path reconstruction

Figure 47: Matching a dynamic number of destination gridlines

Depending on the interpretation of the mouse pointer location as a reference point for the insertion, it might be necessary to run the optimization algorithm in two directions. In any case, gridlines must be matched for both dimensions, vertically and horizontally. This adds a new dimension to the complexity of the matching problem: The relationships between source and destination shapes that are established in one dimension may have an influence on the costs calculated for the other dimension. My approach to this interdependency is to iteratively try a small number of different solutions in either dimension and then decide for the ultimate best guess. This approach does not always yield the actual optimum, but it has the advantage to find a "good" solution in polynomial time.

6 Evaluation

In this chapter I endeavor to evaluate the potential impact of the smart grid concept in general and of my implementation of a user interface in particular. First, I present an interpretation of the measurements taken in the field study (Chap. 2). This leads to an estimation of the over-all performance gain for the target user group. Subsequently, the actual performance of my prototype's user interface is measured for a sample of real life slide layouts and is compared to the working speed of an expert using plain PowerPoint.

6.1 Estimating the Potential Speed-Up

Based on the four partitions of PowerPoint related activities (Sect. 2.2.1), it is assumed that there is no way to speed up *text entry* or *custom drawings*. On the other hand, my thesis claims that there is potential to significantly improve *chart design* and *slide layout*. In this section I estimate the potential speed-up for the latter activities and extrapolate the results to the entire PowerPoint-related workflow.

6.1.1 Speedup for Chart Design

The think-cell add-in for efficient chart design was in beta state at the time of this writing. This software enhances PowerPoint with explicit support for frequently used chart types, which are not supported by Microsoft Graph. Furthermore, it also offers additional UI widgets for direct manipulation. Direct manipulation of charts, which allows user interaction with the graphical chart representation and feeds changes back into the data table, is not provided by Microsoft Graph. Although efficient chart design is not the focus of my thesis, it is an important aspect of PowerPoint usage in business consultancies and exemplifies the speed-up potential of carefully designed user interfaces for computer supported layout. A working prototype for waterfall charts served for a number of case studies that gave an impression of the possible performance gains.

The results are impressive. A non-trivial chart that takes about 12 min to be created by an expert using plain PowerPoint, can be done in about 2 min with the think-cell chart add-in. This translates to a speed-up factor of 6 for the specific task.

6.1.2 Speedup for Slide Layout

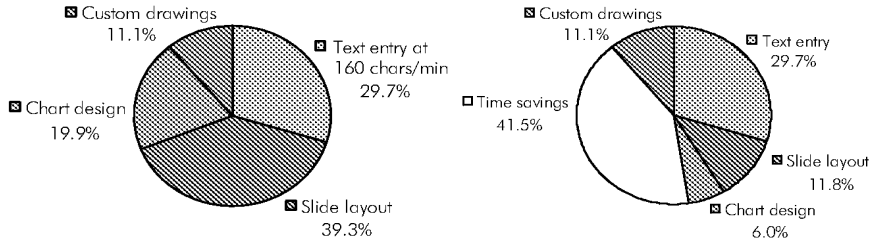
I estimated the potential speed-up of the smart grid approach by simulating the necessary user interaction. The simulation was based on assumptions which subsume the smart grid concept from the user's perspective: The user chooses which element to insert, describes the place in the smart grid where the element should go and in some cases specifies additional constraints.

Without specifying every detail of the user interaction at this point, I could deduce the approximate number of required mouse moves and clicks. For more plausible results, I distinguished *easy* mouse interactions (like selection of a single element) and *difficult* mouse interactions (like multiple paste) and assigned them 2 seconds and 5 seconds time exposure, respectively. I did not take time for typing text into account, but I did account for entering the text entry mode with one easy interaction.

I applied this model to the slide that is shown in figure 9. The expert rating for creating the layout of that slide was 15 minutes. Based on my simulation model, using the smart grid and the solver, the layout could be specified in less than three minutes. This translates to a speed-up factor of 5 which compares well to the factor of 6 that was observed for chart design (Sect. 6.1.1).

6.1.3 Expected Over-All Speed-Up

Based on the findings presented in section 2.2.3, I projected my speed-up estimation on chart design and slide layout to the over-all PowerPoint-related workflow in a business consultancy. For the sake of the argument and to account for the fact that my case studies are not statistically representative, I did not take the speed-up factors of 6 and 5, but rounded off the estimated speed-up to $\frac{10}{3}$. Figure 48 shows the result: In the chosen setting of a business consultancy, a performance gain of about 40 % can be expected for PowerPoint with the think-cell add-in in comparison to the plain PowerPoint.



(a) Current time exposure (comp. Fig. 5(b)) (b) Expected time savings

Figure 48: The estimated time saved with think-cell chart and layout plugins over all stages of the PowerPoint-related workflow is greater than 40 %.

6.2 Case Study

To the running prototype as described in the previous sections, I added some trivial solver functionality that implements "optimal" layout by equidistant gridlines. Hence, the prototype is already manoeuvrable and can serve as an evaluation of the user interface, however the output is far from esthetically satisfying. The underlying assumption is, of course, that with a real numerical solver the resulting layout would appear natural and appealing. In this section, a simple case study is

presented that suggests that the concepts of this work support the speed-up goals of the over-all application.

6.2.1 Question

The ultimate goal of the application is a speedup-factor of about 5 in the PowerPoint-related workflow of a business consultancy. While using plain PowerPoint, the necessity to rearrange elements when text comes into play becomes the major time consumer. Depending on the quantity of text that the different shapes contain, the slide is iteratively refined until all text is readably sized and nicely wrapped, all elements are aligned and have uniformly distributed space between them, and so forth. An increase of efficiency is expected to result from the solver that updates and maintains relations between elements on the slide whenever the user makes some arbitrary addition or modification. With this approach, all the time that is currently spent in iteratively rearranging the slide layout, would be eliminated.

In this scenario, it is important that the user interaction required to specify layout constraints does not require any more effort or time than the user interaction that is currently required to create a pixel-based layout in PowerPoint. It is therefore the job of the user interface and the goal of my thesis to facilitate the constraint-based specification of a certain layout in the same or less time than it takes to specify the same layout using plain PowerPoint.

6.2.2 Setup

To validate the concepts implemented in my prototype, I chose a representative sample of 20 slides from a number of original business presentations. The case study was designed to compare two conditions: The user interface of plain PowerPoint vs. the user interface presented in this work. For each condition, I had an expert copy the layout of those 20 slides from a printout to a PowerPoint slide or smart grid, respectively. In order to measure the impact of the user interface for layout specification as precisely as possible, no text or formatting was applied to the slides. That way, the influence of automatic text wrapping and constraint-maintaining layout was excluded from the measurement and the times taken are exclusively due to user interaction that specifies alignment of shapes.

To measure time for the plain PowerPoint interface, I hired a professional from a business consultancy's visual pool, who was rated by his superior as one of the fastest PowerPoint users in the team. As the only available expert for the new smart grid user interface, the author did the timing by himself. Both experts created the 20 slides in the same order and in one batch.

6.2.3 Results

Figure 49 lists the results from the competition. For $\frac{3}{4}$ of all slides as well as on the average, the smart grid interaction concept was faster to use than the traditional PowerPoint UI. In 3 cases the speedup difference was greater than 60 %. Based on

Slide No	**PowerPoint**	**Smart Grid**	**Difference**	**Percent**
1	01:49	01:19	00:30	27.5 %
2	01:36	01:17	00:19	19.8 %
3	01:05	00:23	00:42	64.6 %
4	03:43	02:36	01:07	30.0 %
5	00:51	01:06	*-00:15*	-22.7 %
6	02:51	01:55	00:56	32.7 %
7	02:59	01:22	01:37	54.2 %
8	00:50	00:45	00:05	10.0 %
9	02:20	01:38	00:42	30.0 %
10	01:18	00:55	00:23	29.5 %
11	01:31	01:21	00:10	11.0 %
12	02:09	02:40	*-00:31*	-19.4 %
13	03:21	01:16	02:05*	62.2 %
14	02:44	03:31	*-00:47*	-22.3 %
15	01:15	01:00	00:15	20.0 %
16	01:01	00:59	00:02	3.3 %
17	01:27	02:22	*-00:55*	-38.7 %**
18	02:09	00:47	01:22	63.6 %
19	01:58	01:05	00:53	44.9 %
20	01:14	01:58	*-00:44*	-37.3 %
Sum	38:11	30:15	07:56	
Avg	01:54	01:30	00:23	
Percent	**100.0 %**	**79.2 %**	**20.8 %**	

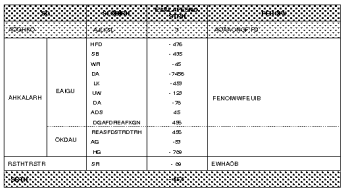
* absolute maximum advantage for the smart grid
** relative maximum advantage for PowerPoint

Figure 49: User interaction times for slide layout specification (min:sec)

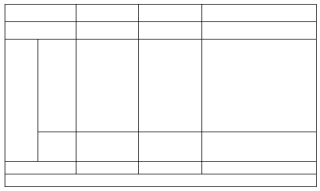
the available data, I can conclude that my prototype reached the goal to allow the specification of constraints for the smart grid in no more time than is necessary to create the same layout in plain PowerPoint. In fact, although I did not collect enough data to make a definite statement, the results of the case study suggest that the smart grid interaction concept might be faster than plain PowerPoint by about one fifth.

However, in 5 out of 20 instances, using the smart grid turned out to be slower than using PowerPoint alone. It will be up to future work to analyze these instances and draw conclusions to further improve the smart grid UI. At this point, we will have a look at two examples that yielded extreme results: With slide 17 PowerPoint was almost 1 minute (39 %) faster than my prototype, while Slide 13 showed a 2-minute-advantage (62 %) for the smart grid (Fig. 50).

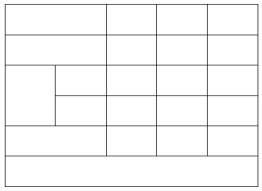
The first thing to acknowledge is that the results produced by the smart grid show only remote resemblance with the original layout. This is due to the fact that at this stage my prototype regards equidistant layout as "optimal". When comparing the outcomes from PowerPoint and from the smart grid, it is only the order of edges that counts, not the actual sizes of the rectangles.



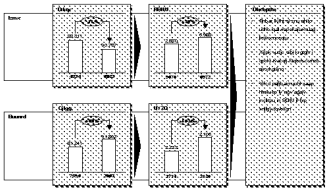
(a) Slide 17 (original)



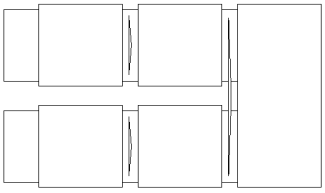
(b) Slide 17 (PowerPoint)



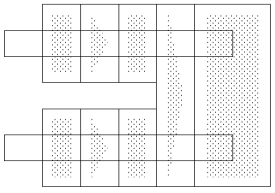
(c) Slide 17 (smart grid)



(d) Slide 13 (original)



(e) Slide 13 (PowerPoint)



(f) Slide 13 (smart grid)

Figure 50: Two examples for tasks and outcomes of the competition: With slide 17, PowerPoint was almost 1 minute (39 %) faster than my prototype, while slide 13 showed a 2-minute-advantage (62 %) for the smart grid.

Moreover, rectangles are the only shapes that are currently available to be inserted by the smart grid user interface. Therefore, where there is an arrow in the original layout, in the layout built by the smart grid you see only a rectangle. Since I suggest that the "smart" implementation of arrows – one of the most frequently used visual elements – have a built-in internal margin, the rectangles you see adequately represent the arrows' placement.

Slide 17 – Advantage for PowerPoint. All edges in slide 17 (Fig. 50(a)–(c)) that mutually depend on each other are collocated and therefore extremely easy to place within PowerPoint's regular grid. On the other hand, the irregularities of this table – some cells span multiple rows or columns – require extra effort to build with the smart grid UI. Although smart gridlines do provide support for precise placement also in this case, the simple one-click insertion heuristics are optimized for a regular table layout. Therefore, the user input required to create this layout involves many drag-and-drop placements and is quite similar to the interaction sequence in plain PowerPoint. Obviously, in this situation the PowerPoint professional outperformed the smart grid expert.

Slide 13 – Advantage for the Smart Grid. The layout of slide 13 (Fig. 50(d)–(f)) contains a lot of symmetry relations that are hard to map to PowerPoint's regular grid. Moreover, each element on the slide is somehow related to all others, meaning that changing the size or position of one element requires adaption of all others. The symmetry relations between non-neighboring edges and the multidimensional interdependency between all shapes on the slide makes this layout especially hard to realize in plain PowerPoint. With the smart grid, it suffices to place rectangles inside or next to other rectangles – assuming that the solver creates a regular and symmetric layout without explicit directions from the user's part.

7 Conclusion

In this chapter I present an outlook of future work that is required or desirable to further pursue the goal of highly efficient slide layout. Finally, a résumé summarizes the lessons learned from my thesis.

7.1 Future Work

During the work on the interaction concept and the user interface prototype, a lot of ideas arose that would be worth further investigation. These ideas, which are briefly discussed in this section, could not be realized in the scope of the present thesis. However, the results from my thesis became part of a PowerPoint add-in for automated layout support, which will continue to be developed and finally will be commercially deployed. Therefore, it is not unlikely that some of the following ideas will contribute to future releases of the software.

Constraints Spanning Multiple Slides. So far, the smart grid concept represents constraints that relate smart elements within a single slide. It is not yet possible to create constraints between elements from different slides. In practice, there is mainly one kind of constraint that spans across multiple slides: Certain elements like headlines and stickers need to be placed in the same location on every slide where they occur. As suggested in section 4.1.2, across-slide constraints could be implemented in a straight-forward manner by *global smart gridlines* that are placed at the same position on each slide.

Styles and Formatting. Although formatting options like fonts and colors are not directly related to spatial constraints, these features play an important role in professional slide design. In particular, consistent formatting of all elements across all slides or even across a set of presentations must be enforced. Style templates as they are known from standard office software address this issue, but exhibit serious usability problems. It would be interesting to see if the concept of constraints could also be applied to styles and formatting, and if it could help to efficiently create consistent designs.

Animated Feedback. As suggested in section 4.2.6, animation could be very effective to explain dynamic layout updates. Further studies with the software will show if users understand what happens to the slide layout when the automation takes over. I would expect that users might get confused when the self-adjusting layout suddenly changes the appearance of a slide. Gleicher *et al.* [Gle92] found that an animated transition between two states of the layout can help to explain the effects of constraints and thus can improve the usability of an automated layout tool.

User Studies with the Prototype. I did a detailed user study for plain PowerPoint (Chap. 2), whereas only a small case study could be conducted on the smart grid prototype (Sect. 6.2). As soon as the prototype can be used to create real-life slides, studies with real users will shed more light on the strengths and weaknesses of the smart grid approach and will help to refine the interaction concept.

Online User Interface Evaluation. In the current PowerPoint add-in, an online error report is used to help finding bugs that occur at the customer's site. A similar feature could facilitate the continued improvement of the user interface: Data about the frequency and the context of the usage of certain features could be collected during productive use of the software. This data would help to refine the software developer's model of the user population and would help to discover usability problems without the need for an intrusive user study. It is understood that a technique like this requires informed consent from the user's part.

Other Application Domains. The smart grid approach could probably be extended to other application domains than business slide layout. For example, the design of graphical user interfaces – which is an active field of research – could benefit from the experiences that were made with the smart grid implementation.

7.2 Résumé

User interfaces for standard software are typically targeted at a large and diverse user population and a broad range of possible tasks. In this respect, Microsoft PowerPoint is a good example: Almost anybody who has used a personal computer before, can create PowerPoint slides with reasonable effort. There is no doubt that the PowerPoint user interface is *effective* for a user population that includes occasional users as well as professionals. However, the *efficiency* for a specialized task, like business slide design, suffers from the attempt to serve everybody well.

The development of efficient user interfaces for a specialized task is relatively expensive: First of all, a solid domain knowledge of the user population and the specific task is required. It takes time and careful analysis to understand an existing workflow and develop appropriate tools. Ideas from the laboratory must be tested against real life conditions. This is only possible by involving real users from the target audience. Moreover, the entire process must be iterated several times before satisfying results can be expected. Under commercial conditions, the cost for this process is often considered too high. As a result, the user interfaces of standard software evolve slowly and not always in correspondence with the actual users' needs.

Highly specialized research systems like Weitzman and Wittenburg's system [WW94] or Graf's LayLab [Gra95] are far too complex to be usable in a professional context. More precisely, the automated layout systems that are developed in research most often explore the potential of automation and do not care much about usability.

In my thesis, I present an approach that is influenced by constraint-based layout automation as seen in research, but at the same time the implementation is based on actual user demands. I spent more than two weeks on-site, watching professional users using some standard software for real-life tasks. Supplementing the observation with interviews, surveys and hands-on experience, I gathered the domain knowledge that was required to come up with an adequate solution. A commercial product in early development state provided the basis to implement a prototype and see how the new interaction concept works in practice. I am well aware of the fact that this work only represents the first iteration in a sequence of others that ought to follow in order to create the optimal user interface for the chosen setting.

A comparative case study suggested that creating initial slide layouts with the new interaction concept is already about 20 % faster than with standard user interfaces. Together with the initial layout, most relevant spatial constraints are implicitly specified and automatically maintained from then on. Due to the fact that the design of business slides is an iterative process, constantly integrating changes and additions into an existing layout, a speed-up factor of around 5 with respect to all layout related tasks appears feasible.

References

[Adoa] Adobe Systems Inc. GoLive Homepage. Website (accessed November 2002). `http://www.adobe.com/products/golive/`.

[Adob] Adobe Systems Inc. Photoshop Homepage. Website (accessed November 2002). `http://www.adobe.com/products/photoshop/`.

[Aut] Autodesk, Inc. AutoCAD Homepage. Website (accessed January 2003). `http://www.autodesk.com/autocad`.

[Axo] Axon Inc. Solo: Visual Report Production Software. Website (accessed November 2002). `http://www.axoninc.com/`.

[Bad97] Alan Baddeley. *Human Memory*. Psychology Press Ltd, revised edition, 1997.

[BBMS99] Greg J. Badros, Alan Borning, Kim Marriott, and Peter J. Stuckey. Constraint Cascading Style Sheets for the Web. In *ACM Symposium on User Interface Software and Technology*, pages 73–82, 1999. `http://citeseer.nj.nec.com/badros99constraint.html`.

[BNB00] Greg J. Badros, Jeffrey Nichols, and Alan Borning. SCWM: An Intelligent Constraint-Enabled Window Manager. In *Proceedings of the AAAI Spring Symposium on Smart Graphics*, March 2000. `http://citeseer.nj.nec.com/badros00scwm.html`.

[Bor90] James Boritz. Of Mice and Menus. Technical Report CS-90-38, Department of Computer Science, University of Waterloo, Waterloo, Ontario, September 1990.

[BS86] Eric A. Bier and Maureen C. Stone. Snap-dragging. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 233–240. ACM Press, 1986.

[CHWS88] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman. An empirical comparison of pie vs. linear menus. In *Conference proceedings on Human factors in computing systems*, pages 95–100. ACM Press, 1988.

[Cor] Michael Cornelison. The Comparison of Menu Selection Time Using Static and Dynamic Menus. `http://citeseer.nj.nec.com/33849.html`.

[DFAB98] Alan Dix, Janet Finlay, Gregory Abowd, and Russel Beale. *Human-Computer Interaction*. Prentice Hall Europe, second edition, 1998. `http://www.hiraeth.com/books/hci/`.

[Fei88] S. K. Feiner. A grid-based approach to automating display layout. In *Proceedings on Graphics interface '88*, pages 192–197. Canadian Information Processing Society, 1988.

[FSM98] Naomi Friedlander, Kevin Schlueter, and Marilyn Mantei. Bullseye! when Fitts' law doesn't fit. In *Conference proceedings on Human factors in computing systems*, pages 257–264. ACM Press/Addison-Wesley Publishing Co., 1998.

[GIM] GIMP org. GIMP: The GNU Image Manipulation Program. Website (accessed November 2002). `http://www.gimp.org/`.

[Gle92] Michael Gleicher. Integrating Constraints and Direct Manipulation. In *Symposium on Interactive 3D Graphics*, pages 171–174, 1992. `http://citeseer.nj.nec.com/gleicher92integrating.html`.

[Gra95] Winfried H. Graf. The constraint-based layout framework LayLab and its applications. In *Proceedings of ACM Workshop on Effective Abstractions in Multimedia, Layout and Interaction*, San Francisco, California, 1995. `http://citeseer.nj.nec.com/graf96constraintbased.html`.

[GW00] François Guimbretiére and Terry Winograd. FlowMenu: combining command, text, and data entry. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 213–216. ACM Press, 2000. `http://citeseer.nj.nec.com/391215.html`.

[HIVB95] Beverly L. Harrison, Hiroshi Ishii, Kim J. Vicente, and William A. S. Buxton. Transparent layered user interfaces: an evaluation of a display design to enhance focused and divided attention. In *Conference proceedings on Human factors in computing systems*, pages 317–324. ACM Press/Addison-Wesley Publishing Co., 1995.

[HKV95] Beverly L. Harrison, Gordon Kurtenbach, and Kim J. Vicente. An experimental evaluation of transparent user interface tools and information content. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 81–90. ACM Press, 1995.

[Hop] Don Hopkins. Pie menu central. Website (accessed November 2002). `http://www.piemenus.com/`.

[Hop91] Don Hopkins. The design and implementation of pie menus. *Dr. Dobb's Journal*, 16(12):16–26, 1991. `http://catalog.com/hopkins/piemenus/ddj/piemenus.html`.

[Hur78] A. Hurlburt. *The Grid*. Van Nostrand Reinhold Company, Melbourne, Australia, 1978.

[IMKT97] Takeo Igarashi, Satoshi Matsuoka, Sachiko Kawachiya, and Hidehiko Tanaka. Interactive Beautification: A Technique for Rapid Geometric Design. In *ACM Symposium on User Interface Software and Technology*, pages 105–114, 1997. `http://citeseer.nj.nec.com/article/igarashi97interactive.html`.

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM Press/Addison-Wesley Publishing Co., 1999.

[JMPS97] R. Johari, J. Marks, A. Partovi, and S. Shieber. Automatic Yellow-Pages Pagination and Layout. *Journal of Heuristics*, 2(4):321–342, 1997. `http://citeseer.nj.nec.com/johari97automatic.html`.

[KB94] Gordon Kurtenbach and William Buxton. User learning and performance with marking menus. In *Conference proceedings on Human factors in computing systems : "celebrating interdependence"*, pages 258–264. ACM Press, 1994.

[LaT] LaTeX Project. Latex: A document preparation system. Website (accessed November 2002). `http://www.latex-project.org/`.

[LF01] Simon Lok and Steven Feiner. A Survey of Automated Layout Techniques for Information Presentations. In *Proceedings of 1st International Symposium on Smart Graphics*, Hawthorne, NY, USA, March 21st–23rd 2001. `http://www.dfki.de/~krueger/sg2001/schedule/lok.pdf`.

[Maca] Macromedia Inc. Director Shockwave Studio Homepage. Website (accessed November 2002). `http://www.macromedia.com/software/director/`.

[Macb] Macromedia Inc. Dreamweaver Homepage. Website (accessed November 2002). `http://www.macromedia.com/software/dreamweaver/`.

[May99] Deborah J. Mayhew. *The Usability Engineering Lifecycle*. Morgan Kaufmann Publishers, 1999. `http://drdeb.vineyard.net/#lifecycle`.

[Mica] Microsoft Corp. PowerPoint Homepage. Website (accessed November 2002). `http://www.microsoft.com/office/powerpoint/`.

[Micb] Microsoft Corp. The Microsoft Windows User Experience: Official Guidelines for User Interface Developers and Designers. Website (accessed January 2003). `http://msdn.microsoft.com/library/en-us/dnwue/html/welcome.asp`.

[Micc] Microsoft Corp. Visio Homepage. Website (accessed November 2002). `http://www.microsoft.com/office/visio/`.

[Mic97] Microsoft Corp. *Microsoft Visual C++ MFC Library Reference*. Microsoft Press, Redmond, WA, 1997.

[OR] Optimoz and RadialContext. Context-sensitive pie menus for Mozilla. Website (accessed November 2002). `http://optimoz.mozdev.org/piemenus/`, `http://www.gamemakers.de/mozilla/radialcontext/`.

[PLVB00] S. Pook, E. Lecolinet, G. Vaysseix, and E. Barillot. Control menus: execution and control in a single interactor. In *Proceedings of CHI 2000*, The Hague, The Netherlands, April 2000. ACM Press.

[SS94] Andrew Sears and Ben Shneiderman. Split menus: effectively using selection frequency to organize menus. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1(1):27–51, 1994.

[Sun02] Sun Microsystems, Inc. Java foundation classes: Now and the future. Whitepaper, last updated Oct-30-2002. `http://java.sun.com/products/jfc/whitepaper.html`.

[Sut63] Ivan Sutherland. *Sketchpad: A Man Machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Technology, January 1963.

[TK95] Mark A. Tapia and Gordon Kurtenbach. Some design refinements and principles on the appearance and behavior of marking menus. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 189–195. ACM Press, 1995.

[W3Ca] W3C (World Wide Web Consortium). Cascading style sheets, level 2 (CSS2) specification. Website (accessed November 2002). `http://www.w3.org/TR/REC-CSS2/`.

[W3Cb] W3C (World Wide Web Consortium). Hypertext markup language (HTML). Website (accessed November 2002). `http://www.w3.org/MarkUp/`.

[WW94] Louis Weitzman and Kent Wittenburg. Automatic presentation of multimedia documents using relational grammars. In *ACM Multimedia*, pages 443–451, 1994. `http://citeseer.nj.nec.com/weitzman94automatic.html`.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt habe und alle verwendeten Quellen und Hilfsmittel im Literaturverzeichnis angeführt habe.

Ort, Datum

(Volker Schöch)